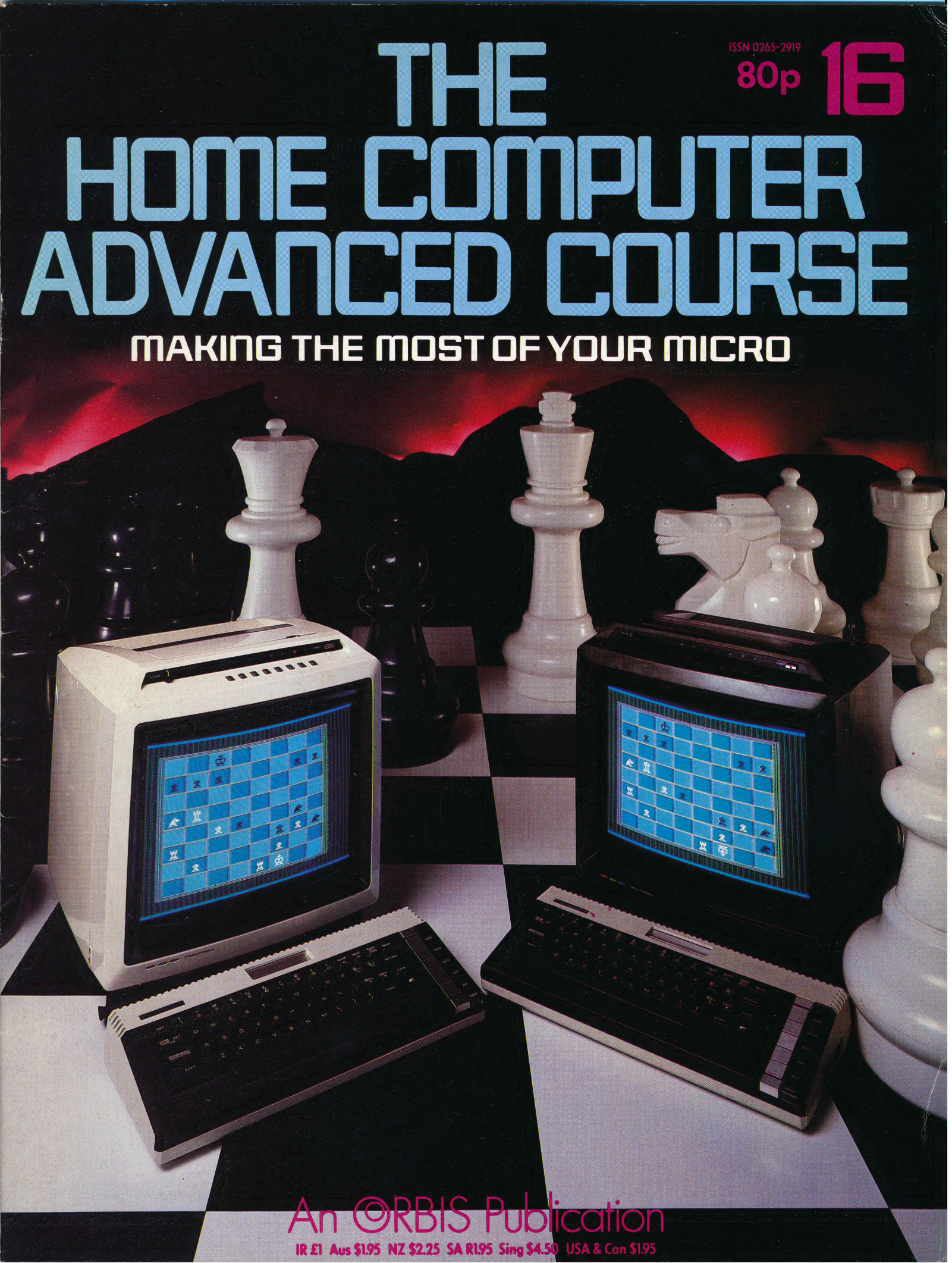
ISSN 0265-2919

80p 16

THE HOME COMPUTER ADVANCED COURSE

MAKING THE MOST OF YOUR MICRO



An ORBIS Publication

IR £1 Aus $1.95 NZ $2.25 SA R1.95 Sing $4.50 USA & Can $1.95

CONTENTS

APPLICATION

CHESS ON CHIPS We look at how one of the most popular board games has been programmed for home micros 301

HARDWARE

GOING DOTTY The first of a short series that takes an in-depth look at dot matrix printers 304

WISH FULFILMENT The Colour Genie is an inexpensive machine for all the family 309

SOFTWARE

IN FORMATION Largely ignored by home micro owners, spreadsheets can be used for a variety of non-financial tasks 306

MINESHAFT MANIA We look at a program that has taken the charts by storm 313

JARGON

FROM COLD START TO COMPILER A weekly glossary of computing terms 308

PROGRAMMING PROJECTS

RIDDLE OF THE SANDS We present a program that poses a sticky problem 312

RUN SILENT, RUN DEEP Our Subhunter project has reached its target 314

MACHINE CODE

THE GREAT DIVIDE We briefly look at how division is programmed, before rounding off this introductory section 316

PROFILE

WELSH RARE BIT Dragon Data is one of the great success stories in the British microcomputing industry 320

Next Week

• We compare some of the more popular television monitors used with home micros.

• Our machine code course begins a new section with an article about programming high resolution graphics on the Commodore 64. In future weeks, we will look at the Assembly languages used by other processors, such as the 6809 at the heart of the Dragons.

• A home computer operating on its own has limitations. By linking into networks, however, a wealth of possibilities materialise.

QUIZ

1) Why do the printer speeds specified by the manufacturers seem faster than those experienced by the user?

2) When a human player uses "brute force" in a game of chess it probably means kicking over the board. What does it mean to a computer?

3) What are the call addresses of the print routines on the Commodore 64, BBC Micro and Spectrum?

4) What connection is planned between the Welsh Dragon and its far eastern cousins?

Answers To Last Week's Quiz

A1) The variable H3 is set to zero and the bit that controls sprite three is reset to zero.

A2) A$(0,0) contains the header record and the number of files held on the tape.

A3) Because calculations are not performed on membership numbers.

A4) The X-axis is restricted by the width of the paper, which limits the plotting range to 480 steps.

QUIZ

COVER PHOTOGRAPHY BY MARCUS WILSON-SMITH

Editor Jim Lennox; **Art Director** David Whelan; **Technical Editor** Brian Morris; **Production Editor** Catherine Cardwell; **Picture Editor** Claudia Zeff; **Sub Editor** Robert Pickering; **Designer** Julian Dorr; **Art Assistant** Liz Dixon; **Editorial Assistant** Stephen Malone; **Contributors** Steven Colwill, Max Phillips, Matt Nicholson, Sue Jansons, Mike Wesley, Geoff Nairn, Martin Hayman, Rose Deakin, Richard Pawson; **Group Art Director** Perry Neville; **Managing Director** Stephen England; **Published by Orbis Publishing Ltd: Editorial Director** Brian Innes; **Project Development** Peter Brookesmith; **Executive Editor** Chris Cooper; **Production Co-ordinator** Ian Paton; **Circulation Director** David Breed; **Marketing Director** Michael Joyce; **Designed and produced by Bunch Partworks Ltd; Editorial Office** 85 Charlotte Street, London W1P 1LB; **© APSIF Copenhagen 1984; © Orbis Publishing Ltd 1984: Typeset by Universe; Reproduction by Mullis Morgan Ltd; Printed in Great Britain by Artisan Press Ltd, Leicester**

HOME COMPUTER ADVANCED COURSE – Price UK 80p IR £1.00 AUS $1.95 NZ $2.25 SA R1.95 SINGAPORE $4.50 USA and CANADA $1.95

How to obtain your copies of HOME COMPUTER ADVANCED COURSE – Copies are obtainable by placing a regular order at your newsagent, or by taking out a subscription. Subscription rates: for six months (26 issues) £23.80; for one year (52 issues) £47.60. Send your order and remittance to Punch Subscription Services, Watling Street, Bletchley, Milton Keynes, Bucks MK2 2BW, being sure to state the number of the first issue required.

Back Numbers UK and Eire – Back numbers are obtainable from your newsagent or from HOME COMPUTER ADVANCED COURSE. Back numbers, Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT at cover price. AUSTRALIA: Back numbers are obtainable from HOME COMPUTER ADVANCED COURSE. Back numbers, Gordon & Gotch (Aus) Ltd, 114 William Street, PO Box 767G Melbourne, Vic 3001. SOUTH AFRICA, NEW ZEALAND, EUROPE & MALTA: Back numbers are available at cover price from your newsagent. In case of difficulty write to the address in your country given for binders. South African readers should add sales tax.

How to obtain binders for HOME COMPUTER ADVANCED COURSE – UK and Eire: Please send £3.95 per binder if you do not wish to take advantage of our special offer detailed in Issues 5, 6 and 7. EUROPE: Write with remittance of £5.00 per binder (incl. p&p) payable to Orbis Publishing Limited, 20/22 Bedfordbury, LONDON WC2N 4BT. MALTA: Binders are obtainable through your local newsagent price £3.95. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Miller (Malta) Ltd, M.A. Vassalli Street, Valletta, Malta. AUSTRALIA: For details of how to obtain your binders see inserts in early issues or write to HOME COMPUTER ADVANCED COURSE BINDERS, First Post Pty Ltd, 23 Chandos Street, St. Leonards, NSW 2065. The binders supplied are those illustrated in the magazine. NEW ZEALAND: Binders are available through your local newsagent or from HOME COMPUTER ADVANCED COURSE BINDERS, Gordon & Gotch (NZ) Ltd, PO Box 1595, Wellington. SOUTH AFRICA: Binders are available through any branch of Central Newsagency. In case of difficulty write to HOME COMPUTER ADVANCED COURSE BINDERS, Intermag, PO Box 57394, Springfield 2137.

Note – Binders and back numbers are obtainable subject to availability of stocks. Whilst every attempt is made to keep the price of the issues and binders constant, the publishers reserve the right to increase the stated prices at any time when circumstances dictate. Binders depicted in this publication are those produced for the UK market only and may not necessarily be identical to binders produced for sale outside the UK. Binders and issues may be subject to import duty and/or local taxes, which are not included in the above prices unless stated.

CHESS ON CHIPS

Chessboard Champ
David Levy is an International Chess Master who resigned from competition against humans in 1978. In 1968 he wagered a large sum on his prowess, betting that no computer chess program would be able to beat him within the next 10 years. Since then, the period covered by the bet has been extended, but he remains undefeated. A leading authority on computer chess, Levy heads Intelligent Software, a company that provides the dedicated programming skills behind many chess computers and microcomputer chess packages. Levy believes that micros are now beginning to approach mainframe performance in chess playing, and estimates that within five to eight years a microcomputer will be able to defeat Belle (a dedicated chess machine) and the mainframe Cray Blitz (which won the 1983 World Computer Chess Championship) — possibly by using parallel microprocessors to speed search functions

One of the most popular applications of home computers has been games playing, and it is not surprising that the game to have received the most attention is one of the oldest and most highly respected of strategic board games — chess. We look at some of the ideas and thinking behind the development of chess programs on computers.

Few games have ever captured the imagination as much as chess: it has been played by millions of people around the world for thousands of years, and it is played today with rules almost unchanged since the 17th century. There are those who devote their lives to the study and mastery of this game of strategy, finding satisfaction in its need for intellectual rigour and agility. The game has spawned a range of variations that attempt to introduce greater levels of complexity: for example, three-dimensional chess involves several boards suspended in space and demands a lot more concentration. Another variant is three-person chess, which is played on a board in the shape of a Y. On the diagonals where the three 'wings' intersect, special rules apply to the movement of the pieces. The theory behind this version is that two of the players will unite against the third and then battle each other for victory. But none of these variations has managed to displace the essential two-person confrontation played out on the 64-square board.

One reason for this is the almost infinite number of variations within the game itself. In 1949, the mathematician Claude Shannon wrote a paper called 'Programming a Computer for Playing Chess', in which he calculated that there are 10^{120} possible games of 40 moves. This means that a person playing chess 24 hours a day, seven days a week, and taking an hour for each game (which isn't long for 40 moves) would take slightly more than 10^{17} years to play all the possible games! Of course, chess has now been so thoroughly analysed that this vast range of possibilities is in practice decreased by a factor that is dependent on the player's experience.

Given this complexity, it is hardly surprising that programming computers to play chess has consumed much time and effort. Chess programs have been run on large mainframe computers for many years, and there are now numerous versions for home microcomputers. The development of high-quality microcomputer chess programs is linked to hardware innovations; problem areas have always been the lack of sufficient memory

Software Strategies

In an effort to evaluate some of the most popular chess programs for home micros, THE HOME COMPUTER ADVANCED COURSE conducted a mini-tournament for these products: Sargon III, running on an Apple IIe (£41.95 disk – Hayden Software, programming by Dan and Kathe Spracklen); Cyrus IS Chess on a 48K Spectrum (£9.95 cassette – Sinclair Software, programming by Intelligent Software); Colossus 2.0 on a Commodore 64 (£12.95 disk – CDS MicroSystems, programming by Martin Bryant); and Grand Master 64, also for the Commodore 64 (£5.95 cassette – Audiogenic, programming by Kingsoft).

Although these programs have played against one another in international microcomputer chess tournaments, we wanted an informal evaluation based on features, playability and competence. The mini-tournament consisted of a minimum of two games for each program, one on the simplest level of play and the second on a higher, competition level. There was no attempt to determine an overall winner.

One of the problems that arises in playing one chess computer against another is that it is often difficult to identify the level of play that gives the two programs a fair and equal footing. Levels are usually defined by the length of time the computer allows itself to search for the best move, but there may not be a direct correlation between a 10-second time limit in one program and the same time limit in another. For instance, Sargon III 'steals' time from its opponent and keeps its move generator operating while its opponent is moving. All the other programs turn off their move generators at this point. Nevertheless, every effort was made to be fair, if not absolutely precise, in pairing the programs.

Quality Of Play

In general, all the programs played sound, if uninspired, chess on the lowest level (taking approximately 10 seconds per move). And all of them made some very strange, apparently useless, moves toward the latter stages of the middle game. This was probably a result of a 'quiet' position, in which the computers simply bided their time until something interesting happened. On the higher, competition level (approximately 10 minutes per move), all four programs showed clever and sometimes brilliant tactical play. The results of the tournament are shown in the chart opposite.

and the relatively low speed of processing, but advances in technology over the last few years have meant that the quality of such programs is now dependent on the software.

As computers are essentially high-speed calculators, computer chess is designed around numerical calculations, which are used to evaluate the two essential elements of the game: the material and mobility. The 'material' of a game refers to the number and strength of the pieces on the board. The chess program allocates each piece a numerical value. The King may be given either an infinite value or an arbitrarily high one, such as 10,000 (this is done because the loss of the King ends the game); the Queen is assigned a value of nine; the rook is worth five; bishops and knights three; and pawns one. When considering whether it is worth sacrificing a piece in order to capture one of the opponent's pieces, the program will compare their values. Most computer chess programs place great importance on relative values, and will rarely swap pieces if this results in a material disadvantage, unless there is a marked gain in positional strength.

'Mobility' is of great importance in chess as any piece is of little value if its movement is restricted. Conversely, its value is enhanced if it can be positioned in such a way that it asserts influence on several locations at the same time. The chess program therefore needs to evaluate mobility as well as the material considerations. In addition, the program must be able to plan ahead, determining the best sequence of moves from any given position. It is here that chess programs can excel, using the speed of the computer to examine a large number of possible moves in a very short time.

Most chess programs use a 'brute force' technique, searching through as many moves as possible in the time allowed. The time allotted for each move is determined by selecting a 'level' of play at the beginning of each game, with each level giving a different time span during which the computer must make a move. These periods vary from a few seconds to several hours, and the longer a computer is allowed to search, the more likely it is to find the best line of attack for the current position.

At each move, the computer determines whether or not the King is in check, and then considers whether pieces are likely to be gained or lost, whether key squares can be occupied, and many other similar questions. The more criteria the program examines, the better the result will be. The final question is to discover if the opponent's King may be forced into a checkmate position.

In games between computers and humans, computers have a distinct advantage in speed and range of search — yet an excellent human player should always defeat an excellent computer chess

Cyrus IS Chess drew with Colossus and defeated Grand Master on the easy level, and drew with Sargon III on the competition level.

Colossus drew with Cyrus IS Chess on the easy level, defeated Grand Master on the competition level, and drew with Sargon III on the competition level.

Sargon III lost to Grand Master on the easy level and drew with Cyrus and Colossus on the competition level.

Grand Master defeated Sargon III and lost to Cyrus IS Chess on the easy level, and lost to Colossus on the competition level.

Feature	Grand Master 64	Colossus	Cyrus IS	Sargon III
Invisible play	NO	YES	NO	NO
List possible moves	NO	YES	NO	NO
Play 'next best'	NO	NO	YES	NO
Replay a game	NO	YES	YES	YES
Show listing	NO	YES	NO	YES
Printout listing	NO	NO	YES	YES
Take back moves	YES	YES	YES	YES
Tutorial	YES	NO	NO	YES
Promote to non-Queen	NO	YES	YES	YES
Human play	NO	YES	YES	YES
Change sides	YES	YES	YES	YES
Analyse problems	NO	YES	YES	YES
Show search	SINGLE PLY	YES	NO	YES
Invert board	YES	YES	YES	YES
Offered draw	NO	NO	NO	YES
Print board	NO	NO	YES	YES
Save game	NO	YES	YES	YES
Disable library	NO	NO	NO	YES
Auto play	NO	YES	YES	YES
Real-time clock	YES	YES	NO	YES

Features

All competent chess programs will include the ability to castle, promote a pawn to a Queen, and capture en passant, and will understand draw and stalemate situations. Some of these programs have very interesting additional features. Sargon III is the program with most extras, and includes a second disk that contains 107 classic chess matches and 45 chess problems. The documentation is superb, with 75 pages in a loose-leaf notebook. Of course, Sargon III was running on an Apple IIe, and was three times as expensive as the other programs. For the money, Cyrus IS Chess and Colossus also offer some very nice features, as you can see from the table.

Conclusion

In terms of playability, Cyrus IS Chess and Colossus are easiest to use because moves are entered by using the cursor, while Sargon III and Grand Master require you to enter the moves in algebraic notation, such as E2-E4 for pawn to King 4. Colossus and Sargon have the best screen displays. Grand Master offers excellent chess for a very low price.

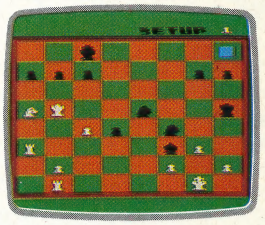
program because of the human's ability to see and create new openings and positions. Computers play superb tactical chess, but, even among human chess masters, a good positional player should beat a good tactical player. Computer chess programmers have focused on tactics because, for a computer, tactical play involves simple number-crunching. If a human opponent makes an unconventional move, the computer will often miss the best response. For this reason, many chess programs have difficulty in dealing with 'quiet' positions, where none of the available moves offer a particular tactical advantage. In these situations, which often occur during the endgame, the program will often simply shuffle pieces around instead of taking the opportunity to plan ahead.

A recently developed style of programming involves 'selective search'. Using this technique, the computer mimics a human player by looking in greater depth at a smaller number of possible moves. Hegener and Glaser, in Germany, have utilised the selective search technique in their Mephisto III program, which looks at every possible move for the first two ply (a 'ply' is one move by one player), then narrows the search and examines a smaller range of moves in depth. Mephisto III also makes an attempt to distinguish between quiet and tactical positions. Techniques of this type should eventually result in computers becoming a real challenge to human players.

The Heart Vs The Head

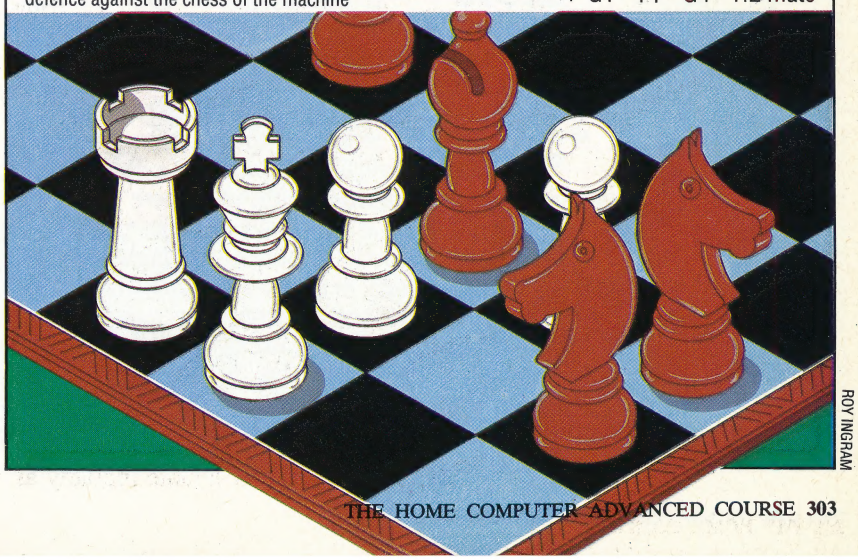
The ability to examine every position up to nine moves ahead almost guarantees chess programs a tactical supremacy over humans. The human chess master's special skill is in selecting a few crucial moves on which to concentrate enormous analytical skill up to 30 moves ahead.

Here, Moritz (black) plays Emmerich (white) in 1922; the position is featured in the film 'Night Moves'. Black can mate by sacrificing his Queen, and then making three very elegant knight moves: most human chess players would unhesitatingly prefer this sequence over all others. Moritz himself missed it and bitterly regretted his oversight. All our packages found mate, but none of them suggested the Knight moves ploy, although some of them must have considered it. The computer's inability to perceive that ending as the 'best' seems to offer humans their only possible defence against the chess of the machine



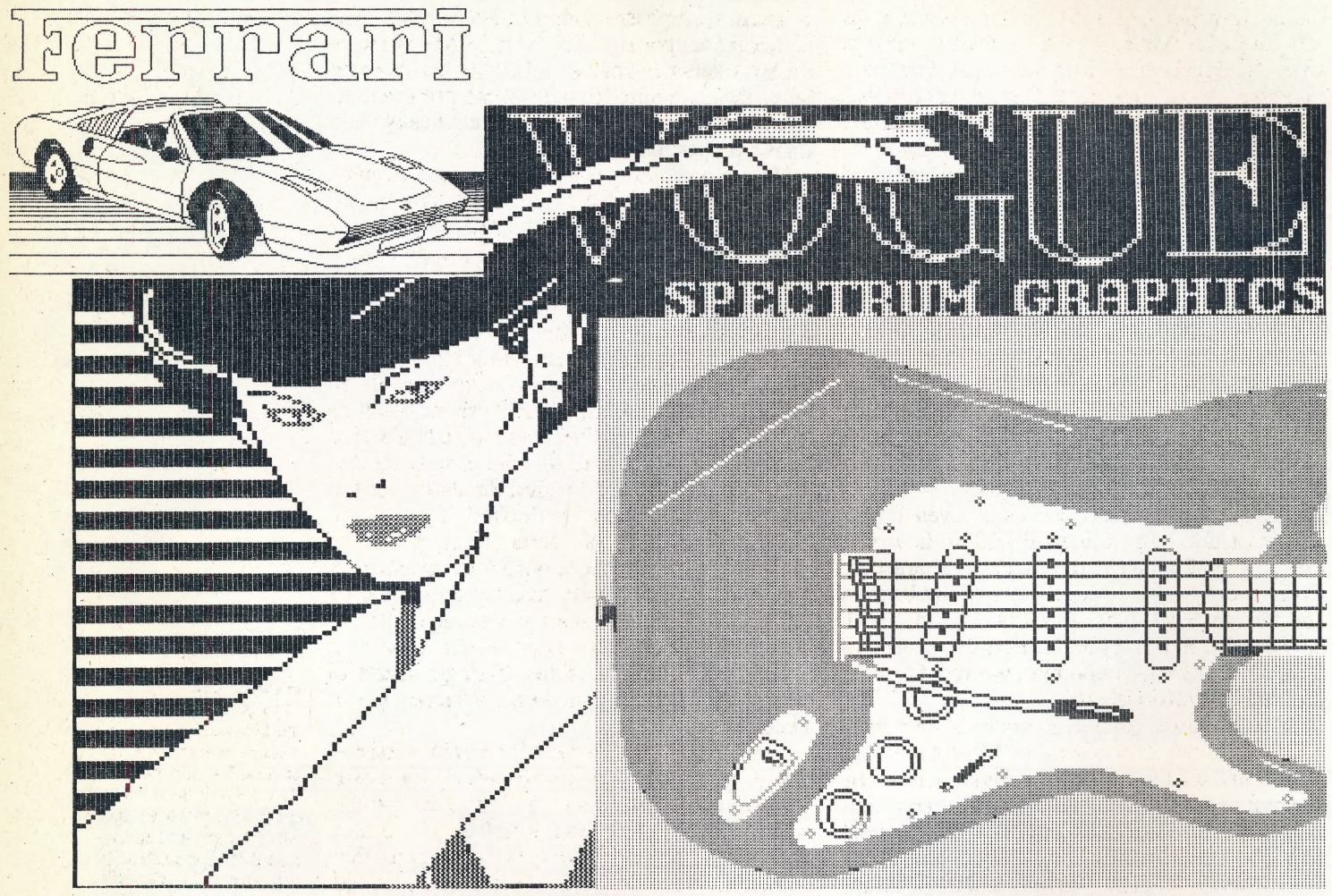
This is the knight moves sequence

1		H5—H2 ch
2	G1—H2	E5—G4 ch
3	H2—G1	F4—H3 ch
4	G1—F1	G4—H2 mate



ROY INGRAM

GOING DOTTY



Printer Artistry
These print-outs show the kind of graphics that can be produced by certain dot matrix printers. Each pin on the print head is controlled individually, and it is possible to produce some complex and satisfying patterns. Details on how to do this will be given in future instalments of the course. These images were created using Paintbox from Print'n'Plotter Products

Most home computer owners eventually decide that the one thing they need to make life complete is a printer. Even if it is used solely for listings, a printer makes a programmer's tasks much simpler — debugging a program is considerably easier if 'hard copy' is available — and a printer is obviously a necessity for word processing.

An inexperienced computer owner is likely to be bewildered by the choice of printers available, as there are almost as many different machines as there are makes of home computer. A decision must first be made as to the type of printer required; this will usually be either a dot matrix or daisy wheel model, although there are other varieties, such as thermal or ink-jet printers. A daisy wheel model produces the highest quality results (generally at a correspondingly high price) and is therefore best for word processing; whereas a dot matrix printer is usually cheaper, faster in operation and ideal for listings and general programming tasks. Here, we will concentrate on dot matrix printers.

A dot matrix printer may be purchased for less than £200, although very sophisticated models can cost £1,000 or more. Important points to consider are the printing speed and the quality of the text produced; more expensive models have extra features such as proportional spacing (i.e. narrow characters such as 'i' are allocated less space than wide ones like 'm') and different character sets. In general, you get what you pay for — you must decide whether such features are worth the extra money.

Printing speed is important as use of the printer 'ties up' the computer because text must be stored in the computer's memory until the printer is ready for it. Therefore, the computer cannot be used for other tasks while printing is taking place. Printer speeds are quoted as 'characters per second' (cps), so whereas an expensive model running at 200 cps might take one minute to print out a long program listing, a cheaper model with a print speed of 30 cps would take more than six minutes to produce the same listing — and during that six minutes the computer cannot be used for any other tasks. This problem may be overcome by using a printer *buffer*. This is simply a circuit board containing RAM chips, which is connected between the printer and the computer and stores the data while

the printer works on it, thus freeing the computer for other operations. More expensive printers have large buffers built in.

The print speeds quoted by the manufacturers should, however, be taken with a pinch of salt. As with car fuel consumption figures, these are always given for ideal conditions and often bear little resemblance to real life! Printer speeds are calculated for the production of a single line of text composed of the same character. Normal text, with its different characters, spaces, line feeds and carriage returns, slows down the print head. Thus, a printer with a quoted speed of 160 cps would probably average only about 100 cps when printing out a program listing.

The quality of the characters produced on the paper varies considerably from printer to printer. It depends mainly on how many pins are used in the print head — the mechanism that forms the characters on the paper. The cheapest models use just seven pins in the print head, whereas the more expensive machines can have 16 or more. On the Commodore printer, which has only seven pins, the characters are produced as a seven by six matrix of dots. The Canon PW1080, however, uses a 16 by 23 matrix to produce its characters. Consequently, the individual dots cannot be seen and the characters have a clearly defined, 'solid' appearance. For program listings, the quality of the print is not really important; whereas for word processing it obviously is.

A dot matrix printer is really a dedicated microcomputer; it uses ROM and RAM memory chips and has a microprocessor. As such, it can be programmed to do other things apart from printing text. This is done by sending special control codes from your micro to the printer, or by setting small switches — known as DIP (Dual In-line Package) switches — inside the printer case. For example, the standard ASCII character set, which is stored in the printer's memory, can be altered to suit different alphabets. In Britain, the hash sign (#) is often changed to print as a pound sign (£).

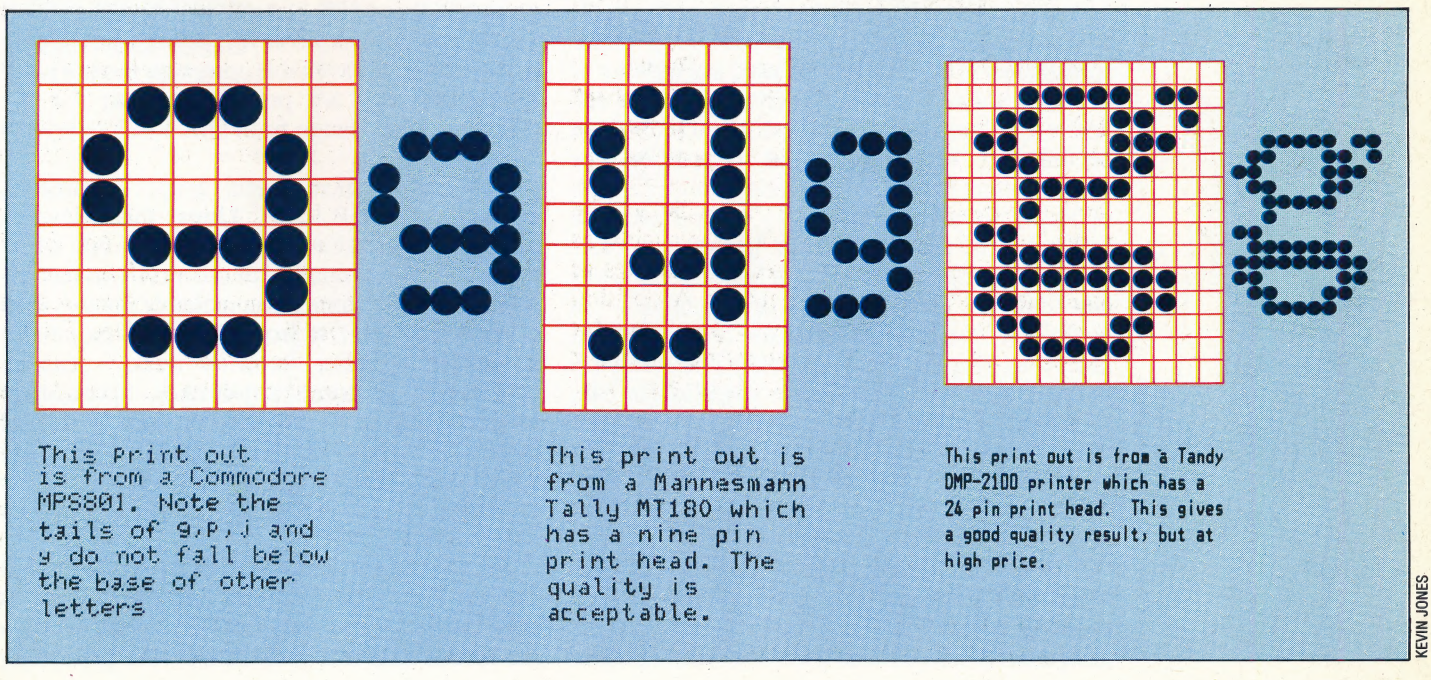
Other special effects include double-width characters, emphasised (darker, heavier) text, and different line spacings. The Epson FX80 is one of the more versatile dot matrix printers and has over 70 of these printing features. It can print in italic characters, underline text automatically and allows proportional spacing.

The Epson range of printers has become something of an 'industry standard'. This means that much of the software that requires a printer — word processing packages, invoice programs, etc. — assumes you have an Epson. This is an important point, for the different makes of printer are by no means compatible.

Other considerations may well influence the choice of printer; certainly, reliability is an important factor. A cheap £200 printer might be all right for producing the occasional listing but it is unlikely to stand up to the continual daily use that an office printer would suffer. Similarly, noise is one factor that is often overlooked: if you like to burn the midnight oil, some printers can be positively deafening at one o'clock in the morning. Does it have a friction feed? All dot matrix printers come with a 'tractor' feed, which will work only with continuous paper — the type with sprocket holes up the sides. If single sheets of paper must be printed, however, a friction feed is necessary.

Finally, perhaps the most important factor — will it work with your micro? Most dot matrix printers come with either a Centronics parallel socket or an RS232 serial interface. If a printer does not have the right one for your micro, then sometimes an alternative interface can be fitted, although this can add over £50 to the price. Even with the right interface, the correct cable is needed to connect the printer to the computer.

Pinprick Details
These print samples show the difference in quality between several dot matrix printers. The main reason for the variation is the number of 'pins' in the print head; those with the most pins have the most detailed characters. The first sample uses only seven pins, and can't produce the 'tails' of the letters g, p, q and y below the line. It is said to lack 'true descenders'



This print out is from a Commodore MPS801. Note the tails of g, p, j and y do not fall below the base of other letters

This print out is from a Mannesmann Tally MT180 which has a nine pin print head. The quality is acceptable.

This print out is from a Tandy DMP-2100 printer which has a 24 pin print head. This gives a good quality result, but at high price.

KEVIN JONES

IN FORMATION

The spreadsheet was one of the first major applications for microcomputers. Its use in the home, however, has been hampered by the assumption that it is strictly for business use. But the spreadsheet has a range of useful possibilities, serving as an 'ideas generator' as well as a valuable aid in the collation and display of information.

Like a word processor or database, the spreadsheet has many facilities that are rarely explored by its users. Most people with a word processing system seldom use its more sophisticated commands, while database programs tend to be used as file management and index systems to the exclusion of their data processing abilities. The majority of home micro owners, however, don't own a spreadsheet program and can't see the need for one. Many believe they would find such a program boring and of little practical use, and are generally intimidated by its association with financial and business uses. This view of spreadsheets underestimates the importance of financial management in the home, and overlooks the fact that spreadsheets are simply ideas processors that have become stuck with the accountancy image. In fact, as word processing is to text, spreadsheets can be to concepts.

A spreadsheet is really a text editor and calculator in one. It is called a spreadsheet because it is divided into rows and columns like an accounting spreadsheet, with data shown in cells, or boxes. These are like the cells of a paper spreadsheet in that they can be used in a number of different ways: text can be entered into a cell where it will remain on display, numerical data can be entered for display and calculation, or mathematical formulae operating on the contents of other cells can be entered. Once some formulae are in place, the spreadsheet becomes a user-generated program waiting for input. Whenever new data — text, numeric, or algebraic — is entered, all the formulae cells are recalculated in turn, thus keeping the spreadsheet constantly up to date with data input. The spreadsheet can, therefore, be used for simple screen/printer layout tasks, making it easy to format and to print not only calculations you could do yourself (if they weren't so tedious), but calculations you would never otherwise have thought of.

In many cases, using a spreadsheet will help to reveal needs the user was unaware of — such as keeping inventories, analysing sports results, designing forms, producing timed synchronisation charts for theatre sound and lighting cues, generating tax returns, deciding whether to rent or buy a television, and so on. All these things could be programmed by someone with a working knowledge of BASIC, but each would take hours to develop, and most of this time would be spent on working out and debugging the endless PRINT TAB, PRINT AT, and INPUT commands needed for formatting the screen display. The great advantage of the spreadsheet is that you format the display as you work out the relationships between your variables. This is done as naturally as you would lay out a sheet of paper, by writing the text, data and the results of calculations wherever you want them to be on the display.

Spreadsheets support a variety of commands to make layout easy: you can copy, move or delete blocks of cells, insert and delete rows and columns, and define the format of a cell or block in terms of size, justification (alignment with other items in the same column), and position of the decimal point. These are exactly the details that are so difficult to handle in most dialects of BASIC, but which are vital to the appearance and ease of use of any report.

Analogous to these formatting facilities are the calculating functions. With a single command you can calculate the mean value of a row or column of data, count the non-zero entries in a table, work out the sum of an array of values, find the maximum and minimum values in a list, and use these facilities in mathematical expressions with more familiar operators and functions such as '+' and '/', SQR and ABS. Not all spreadsheets support all of these facilities, however. The options offered depend upon the available computer memory and how much you are prepared to pay for the program. Prices range from around £5 to several hundred pounds.

Perhaps the most useful single spreadsheet command is REPLICATE. Using this allows a calculation or value typed into one cell to be duplicated in any number of other cells, so that the setting-up of accumulating tables of data — such as mortgage interest from month to month, or household spending week by week — can be achieved in a dozen key-strokes. Spreadsheet programming very quickly becomes a natural extension of arithmetic BASIC, enabling complicated mathematical expressions to be expressed in a more straightforward way than BASIC allows.

Completed spreadsheets can be SAVEd to and LOADed from tape and disk, and many versions offer the option of saving just the text and data in a file format that can be used by word processing

Marking Time

Format
The FORMAT command has been used to set the width of column D, left-justify all text cells, and display all numbers to two decimal places

Copy
Any block of cells can be copied to any part of the sheet by the COPY command

Scaling Factor
Multiplied by an actual mark to produce a corresponding scaled mark

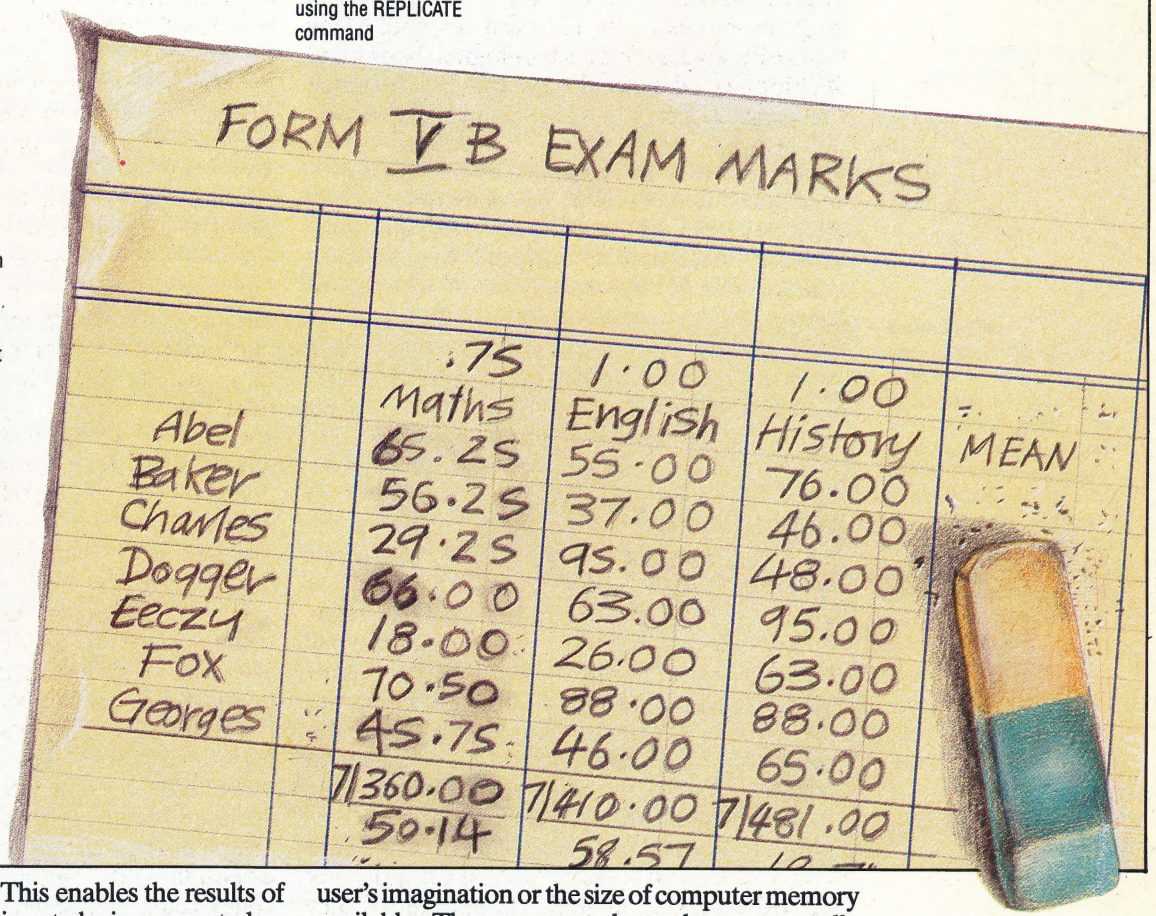
```
   | C  |D| E  || F  || G  || H  |  | J  || K  || L  || M  |
 1| MARKS ADJUSTMENT EXAMPLE
 2|*****************************************************************
 3|                ACTUAL    MARKS          *        SCALED    MARKS
 4|*****************************************************************
 5|                                         *   .75     .86     .73
 6|          Maths    Eng    Hist    MEAN   *  Maths    Eng    Hist    MEAN
 7|Abel    :  87.00  55.00  76.00  72.67   *  65.25  47.30  55.48  56.01
 8|Baker   :  75.00  37.00  46.00  52.67   *  56.25  31.82  33.58  40.55
 9|Charles:  39.00  95.00  48.00  60.67   *  29.25  81.70  35.04  48.66
10|Dogger  :  88.00  63.00  95.00  82.00   *  66.00  54.18  69.35  63.18
11|Eezy    :  24.00  26.00  63.00  37.67   *  18.00  22.36  45.99  28.78
12|Fox     :  94.00  88.00  88.00  90.00   *  70.50  75.68  64.24  70.14
13|George  :  61.00  46.00  65.00  57.33   *  45.75  39.56  47.45  44.25
14|          ============================  *  ============================
15|  MEAN  :  66.86  58.57  68.71  64.71   *  50.14  50.37  50.16  50.23
16|                                         *
17|*****************************************************************
18|
```

Repeating Text
A single star typed into this cell fills the whole row through the REPEAT TEXT feature

Autocalc
Once the formula for one cell is entered, it can be copied automatically to other cells using the REPLICATE command

Mean
Calculated by the AVERAGE(cell#1:cell#2) command

To compare the performance of his pupils in different subjects, the teacher wants to scale all the exam results so that the mean mark in each subject is the same. He has to experiment with different scaling factors for each subject, calculating and recalculating the marks, which is tedious error-prone work that a spreadsheet could do in minutes. On the computer spreadsheet everything except the actual marks is calculated automatically; changing the scaling factor, for example, produces a complete new column of scaled results for that subject in seconds



and database software. This enables the results of calculations and projections to be incorporated *en bloc* into a text or data file, and is a valuable step towards integrated software. This usually applies only to the more expensive packages.

Given a reasonable set of commands, a spreadsheet program is limited mainly by the user's imagination or the size of computer memory available. The programs themselves are usually extensive, and applications with large tables and sophisticated data processing facilities can quickly fill the rest of memory. Complicated calculations, moreover, can appreciably slow the program's calculating response.

COLD START

A *cold start* is what happens when you switch your computer on, no matter whether it is one second or several days since it was last in use. The individual circuits and chips will take an appreciable time (though still measured in terms of a fraction of a second) to stabilise, during which their behaviour is totally unpredictable. Therefore, a simple circuit (consisting of little more than a resistor and a capacitor) is incorporated, which is sure to produce a pulse perhaps a tenth of a second after the machine is turned on. The output of this circuit is connected to the reset line of the microprocessor, which will have stabilised by the time the pulse is applied.

The display screen of a Commodore 64 after it has been cold started

A signal on this line always causes the micro to abandon whatever it is doing (which will be garbage) and go to a piece of code in a predetermined area of ROM, called the *cold start routine*. This routine usually checks through all areas of RAM to establish how much is available and that it is functioning correctly. It will then jump to the operating system, or boot it in from disk to RAM. A *warm start*, by contrast, is what happens when you press the Reset button on your computer. The warm start routine doesn't perform the memory checks, as this would obliterate the contents. Instead, it merely resets the microprocessor registers and jumps back to the operating system.

COMMAND LANGUAGE

We have already explained the difference between a package that is menu-driven and one that is command-driven: the former presents you with a list of options that can be selected with a single key-press, while the latter requires you to type in a command word at the bottom of the screen for each new action. The menu system has advantages for the newcomer to computing, but a command-driven program may in addition feature a command language.

A *command language* gives the user the ability to combine several individual commands into a single operation. It is common in database applications to find yourself repeating sequences like this: GET the next record, EXTRACT the TOTAL field, MULTIPLY this by the DISCOUNT rate, UPDATE the record with the new value, and STORE the record back in the file. A command language enables such a sequence to be activated by a single command such as MODIFY.

The more sophisticated database packages, of which Ashton-Tate's dBase II is the best example, take this a stage further and allow whole applications to be written in the command language, complete with conditional statements and subroutines. The end result is really a programming language just like BASIC or PASCAL, but using more sophisticated commands. Thus writing an information retrieval application in dBase II language will require considerably less work than writing it in a conventional language.

COMPARATOR

A *comparator* is an electronic circuit featuring two analogue voltage inputs and a single output. Its function is to compare the two inputs so that the output will be in one predetermined state if A is greater than B, and in another state if B is greater than A.

One method of building an analogue-to-digital converter is to use a digital-to-analogue circuit (which is much simpler to construct) in conjunction with a comparator. The D/A is connected between the computer and one of the comparator's inputs, with the analogue signal to be measured connected to the other. The computer then generates a counting sequence in binary, which causes a gradually rising analogue voltage at the output of the D/A. When the computer-generated voltage reaches the level of the signal to be measured, the output of the comparator will change, indicating to the computer that its current binary value is the digital equivalent of the analogue signal.

COMPILER

A computer program that translates a program written in a high-level language (called the source code) into a lower-level language (called the object code) is called a *compiler*. At the end of the compiling operation, the user's program will exist in two forms: a *source* file and an *object* file.

Programs written using a compiler are much faster than programs written with the BASIC that comes with home computers. This is because most versions of BASIC are *interpreters* rather than compilers. Interpreters are languages that convert each program instruction into machine code, one at a time, while the program is running. This means instructions inside a loop will be repeatedly converted into machine code, which wastes time.

Compilers, on the other hand, convert all instructions into machine code before the program is used and store them. Thus, the program wastes none of its time converting program instructions into machine code. Compilers are rarely used on home computers because they are complicated and need lots of memory. The few that are available tend to be limited in what they can do.

WISH FULFILMENT

The Eaca Colour Genie is a large, sturdy machine that is designed for home use. Its robust casing contains many features that are unusual in machines costing less than £200. These include an internal power supply, an on/off switch with indicator LED, a built-in aerial lead and several peripheral interfaces.

Based around the popular Z80 microprocessor, the Colour Genie boasts a typewriter-style keyboard with 62 keys. These include four function keys, two Resets (which must be pressed simultaneously) and a Mode Select key, which allows pre-defined graphic characters to be obtained from the keyboard.

The machine has 32 Kbytes of memory. Of this, two Kbytes are set aside for system use, and high resolution graphics use a further four Kbytes. The 16 Kbytes of ROM contain an extended version of Microsoft BASIC, which offers none of the structured programming features found in more recent dialects of BASIC. However, it does allow integer variables, single and double precision variables, multi-dimensional arrays of any variable type, and extensive string-handling facilities. It includes many useful commands to handle sound and high resolution graphics.

Sound facilities are relatively sophisticated, offering three channels (allowing chords to be played), and providing output through the television set. Two BASIC commands control sound generation — PLAY gives a pre-defined sound similar to a glockenspiel, while SOUND allows other noises to be generated.

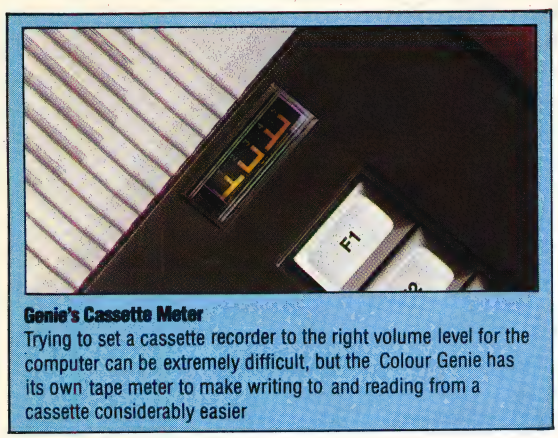
Although extensive and powerful, the Colour Genie's graphics facilities are now somewhat outdated. The screen is considered as two 'pages' (really two different areas of screen memory), one of which stores and displays text, graphics character blocks and user-defined graphics characters, while the other page is used for the display of high resolution graphics. In text mode, the Genie can display up to 25 lines of 40 characters. In graphics mode, the display size is 160 × 102 pixels — which is hardly 'high resolution' by current standards.

GRAPHICS MANIPULATION

The Mode Select key accesses the high resolution page, setting aside 4 Kbytes of memory. The BASIC features numerous commands for graphics manipulation — you may draw lines, fill areas with solid blocks of colour and define, draw and erase shapes. When incorporated into a BASIC program, the command FGR displays the graphics page, but the computer will automatically revert to text

CHRIS STEVENS

Bashful Genie
One computer that has never achieved the fame of the Spectrum or the Commodore 64 is the Colour Genie, even though it's been around for just as long. All the same, it has a small but dedicated following. The machine has 32 Kbytes of memory and unusual joysticks — they come in a matched pair with built-in number pads and their own holder — for £50



Genie's Cassette Meter
Trying to set a cassette recorder to the right volume level for the computer can be extremely difficult, but the Colour Genie has its own tape meter to make writing to and reading from a cassette considerably easier

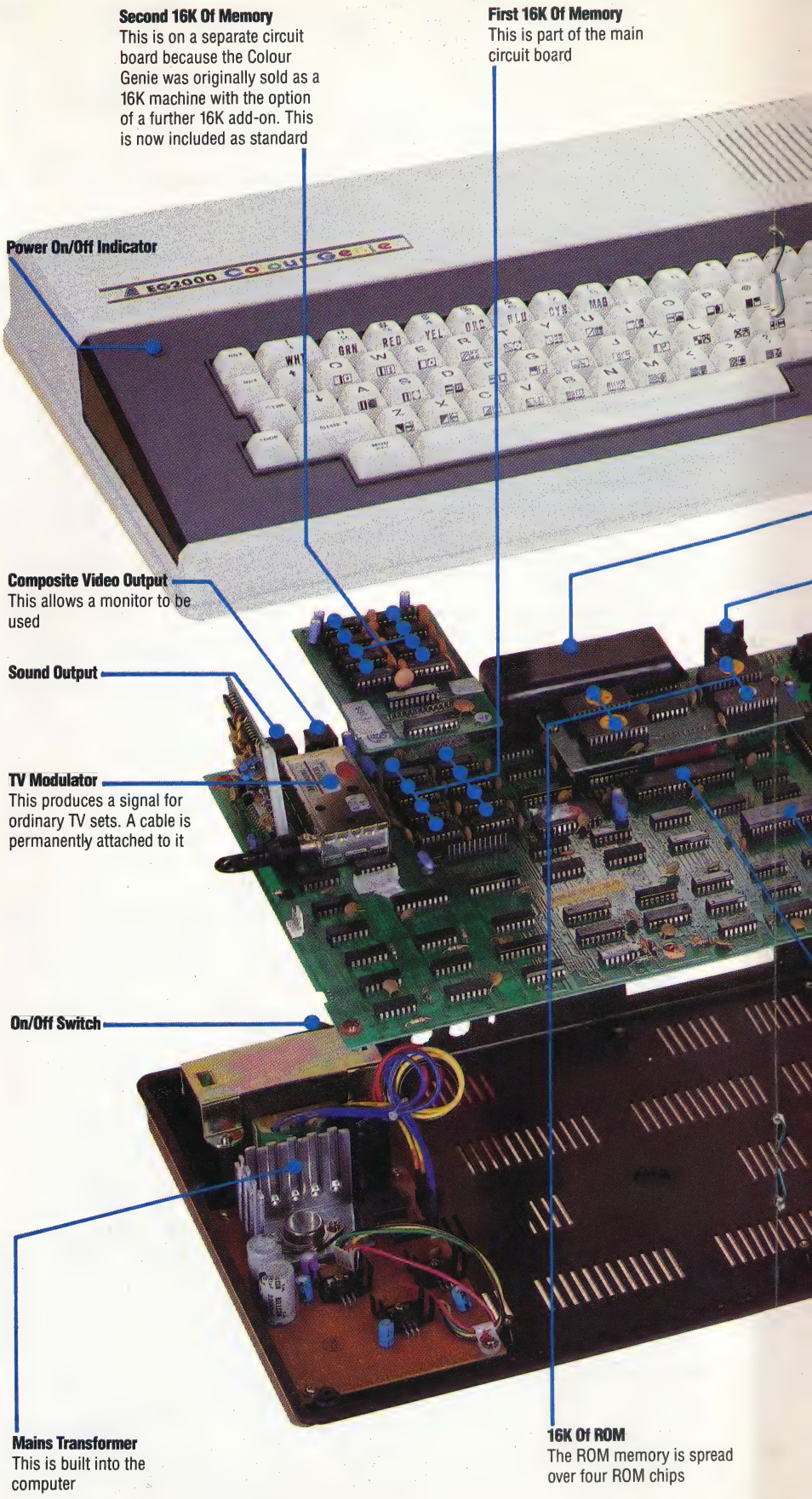
mode at the end of the program. The BASIC also includes commands to clear the graphics page (FCLS), and alter background (FILL) and foreground (FCOLOR) colours. This system is clumsier than the single-page arrangement adopted by most new machines, but it does allow each pixel to be individually coloured (unlike the Spectrum, for example, which has a higher resolution but limits the colours that may be displayed within each eight by eight pixel block). Most arcade-type software uses the text screen for speed, with user-defined characters to give a high resolution effect.

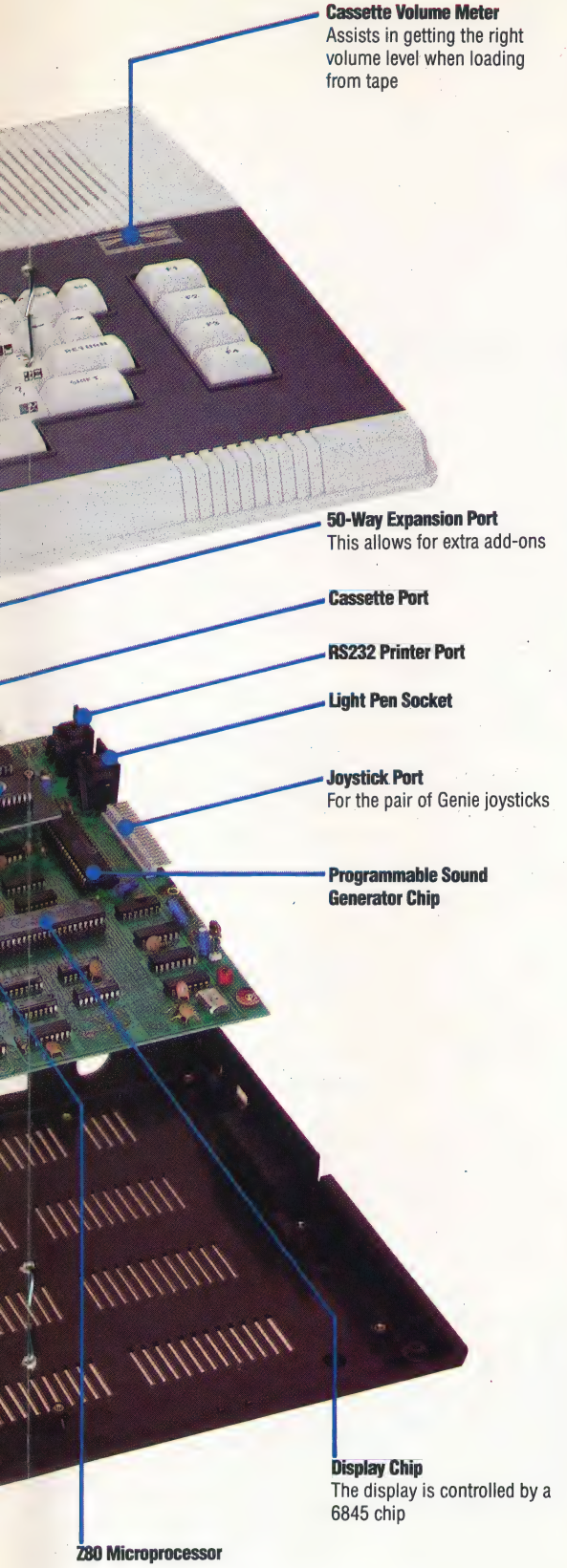
The screen display is clear and steady, but the character set used makes text a little difficult to read. The Genie offers eight colours — white, red, green, yellow, cyan, magenta, blue and orange — all of which may be displayed on the text screen at the same time. High resolution graphics restrict the user to four colours (red, blue, green and black) but there is an additional command (BGRD) to set the graphics page background to pink.

Several interfaces are included: an RS232 port for printers and modems; a 50-way expansion port, which is used for connecting disk drives; a composite video output; an audio output; a light pen socket and a joystick port. Available peripherals include joysticks, a Centronics printer interface, a Prestel cartridge (which requires a modem) and disk drives. The dual joysticks feature built-in keypads but are difficult to use — a lot of pressure is needed to make them respond, and the joysticks do not return to the central 'neutral' position when the pressure is released. Eaca, the company that makes the Colour Genie, does not offer a disk drive for the machine. One is available thanks to a British company that offers its own disk drive using an operating system called QDOS, similar to Tandy's TRS-DOS.

A recording-level meter is built into the case to counter cassette loading problems; the user simply adjusts the volume until the needle is centred, and cassette tapes should then load easily. In addition, a 'data stabiliser' may be fitted between the cassette player and the Genie's cassette lead; this 'cleans' the signal and also aids in tape operation.

The Colour Genie is supplied with two manuals — a beginner's guide and a basic manual. Both are clearly written but are lacking in detail, and neither


Second 16K Of Memory
This is on a separate circuit board because the Colour Genie was originally sold as a 16K machine with the option of a further 16K add-on. This is now included as standard
First 16K Of Memory
This is part of the main circuit board
Power On/Off Indicator
Composite Video Output
This allows a monitor to be used
Sound Output
TV Modulator
This produces a signal for ordinary TV sets. A cable is permanently attached to it
On/Off Switch
Mains Transformer
This is built into the computer
16K Of ROM
The ROM memory is spread over four ROM chips


Cassette Volume Meter
Assists in getting the right volume level when loading from tape
50-Way Expansion Port
This allows for extra add-ons
Cassette Port
RS232 Printer Port
Light Pen Socket
Joystick Port
For the pair of Genie joysticks
Programmable Sound Generator Chip
Display Chip
The display is controlled by a 6845 chip
Z80 Microprocessor

has an index. In fact, the basic manual doesn't even have a contents page. For more advanced users, a technical manual is available at extra cost.

Despite its old-fashioned appearance, the Colour Genie appears to offer good value for money. It falls firmly into the 'home user' category, and has little to offer the scientific or business user. The robust construction, good sound capabilities, full range of peripherals and fairly standard BASIC should make this machine especially attractive to the beginner.

Martian Raider

IAN McKINNELL

Software Choice
The availability of software for the Colour Genie is fairly limited, but the quality of what can be obtained is generally very good. Most of the software is games, and these are often translated versions of games produced for the better known machines

COLOUR GENIE

PRICE
£168

DIMENSIONS
90 × 280 × 340 mm

CPU
Z80, 2.2MHz

MEMORY
32K RAM, 16K ROM

SCREEN
Up to 25 rows of 40 columns of text, graphics up to 160 × 102 pixels. 8 colours in text mode; 4 colours for graphics

INTERFACES
Joysticks (2), RS232 port, light pen port, expansion port

LANGUAGES AVAILABLE
BASIC included

KEYBOARD
Typewriter-style with 62 keys, including four function keys

DOCUMENTATION
The machine is supplied with a beginners guide and basic manual. Both are clearly written, but are too brief to be much use, and neither has an index. A technical manual is also available

STRENGTHS
The Colour Genie is a good 'family' machine. It is sturdily made, uses Microsoft BASIC, offers reasonable graphics and good sound, output via the television

WEAKNESSES
The Genie's design is old-fashioned — it has a slow processor and the screen is handled as two 'pages'. There is little software available

Colour Genie Joysticks
The Colour Genie's joysticks are very attractive, but expensive and difficult to use. A lot of pressure is needed to move the sticks and they do not centre themselves properly after being moved. The built-in number pads are unusual but not very useful

CHRIS STEVENS

RIDDLE OF THE SANDS

Not all games are about shooting aliens. Some, such as the program we present here, demand logical thinking. Trying to drive a truck across a desert is not easy when it can't carry enough fuel to get it all the way. Solving the problem requires a little experimentation or even another program!

Our game is set in a desert 1,000 kilometres wide. Every hundred kilometres or thereabouts is a staging post where fuel tanks may be stored. Back at base, you've as many petrol tanks as you could possibly need, each one large enough to fuel the truck from one stage to the next. The journey across the desert would be quite simple apart from one thing; the truck has room for a maximum of eight tanks at any one time. Therefore, to make it across the desert, you must build up supplies of fuel at various points on your route, driving back and forth between them.

Obviously, the first objective of the game is to make sure that you don't run out of petrol — it's a long walk back to base and the desert isn't a place for a pleasant stroll. Secondly, you must complete the journey travelling the shortest distance and using the fewest tanks possible. You should find this relatively easy to solve with the program set up for eight tanks.

However, we can alter the game to make the problem a little more mind-stretching. What happens, for example, if you can carry only four or six tanks at a time? To investigate these variations, you must alter the value of the variable M in line 60 and try the problem again. You should discover that you are using the same technique but that the intervals between your fuel dumps and the number of journeys made are altered. Can you devise an algorithm that is certain to see you safely across the desert every time? Such an algorithm could be the basis for a program that *solves* this particular problem.

Our puzzle demonstrates an invaluable technique for solving problems in the development of a program. You must first experiment with the information given, try lots of worked examples and then, if all goes well, discover an emerging pattern. From this you can devise an algorithm and then come up with a program. If you want to develop our Desert Trucker game, you could add graphics and other refinements, introducing difficulties like needing to carry water as well as fuel.

Basic Flavours

This program is written in Microsoft BASIC, so should run unchanged on most machines with a 40 × 25 screen display. Spectrum owners must insert LET before all assignment statements.

CHR$(26): Replace by CLS on Spectrum, Oric-1, Atmos, Dragon and BBC; and by CHR$(147) on Commodore 64 and Vic-20.

MID$(STR$(A(1),2)): Replace by STR$(A(1)) on the Spectrum and any machine on which the command PRINT LEN(STR$(2)) produces the result 1.

THEN 1260 & THEN 1300: Change to THEN GOTO 1260 & THEN GOTO 1300 on the Spectrum.

```
10 REM **** Desert Truck
30 DIM A(10)
40 A(1)=80: REM Tanks at start
50 S=1:REM Truck's position
60 M=8: REM Max. tanks on truck
70 N=0: REM Total tanks used
100 REM Next turn
110 PRINT CHR$(26): REM Clear Screen
120 PRINT "STAGE:  1  2  3  4  5  6  7  8  9  10"
130 PRINT "TANKS: ";
140 FOR I=1 TO 10
145 A$=MID$(STR$(A(I)),2)
147 IF LEN(A$)<2 THEN LET A$=" "+A$:GOTO 147
150 PRINT A$;" ";:NEXT I:PRINT
160 X=9+(S-1)*3:PRINT TAB(X);"^^";PRINT TAB(X);"|";:PRINT TAB(X-2);
"TRUCK":PRINT  TAB(X-1);T
175 IF S=10 THEN PRINT:PRINT"You did it!!!":GOTO 1400
180 IF T=0 AND A(S)=0 THEN PRINT : PRINT "Oh dear ...
you d better start walking":GOTO 1400
190 PRINT:PRINT"Your options are :":PRINT
200 IF T>1 THEN PRINT "D Drop fuel tanks"
210 IF A(S)>0 THEN PRINT "F Forwards one stage"
220 IF T>0 AND S>1 THEN PRINT "B Backwards one stage"
230 IF T>0 THEN PRINT "P Pick up fuel tanks"
240 PRINT:PRINT "Which";:INPUT A$
250 IF T>1 AND (A$="D" OR A$="d" )THEN 1260
260 IF A(S)>0 AND (A$="P" OR A$="p") THEN 1300
270 IF T>0 AND (A$="F" OR A$="f") THEN S=S+1:T=T-1:N=N+1:GOTO 110
280 IF T>0 AND S>1 AND (A$="B" OR A$="b") THEN S=S-1:T=T-1:N=N+1:
GOTO 110
290 GOTO 110
1250 REM Drop tanks
1260 PRINT:INPUT "How many ";A
1270 IF A>T OR A<>INT(A) OR A<0 THEN PRINT:PRINT"Please try again
";:GOTO 1260
1280 A(S)=A(S)+A:T=T-A:GOTO 110
1290 REM Pick up tanks
1300 PRINT:INPUT "How many ";A
1310 IF A>A(S) OR A<>INT(A) OR A<0 THEN PRINT:PRINT "Please
try again":GOTO 1300
1320 IF A+T>M THEN PRINT:PRINT"Sorry - only room for";M;"tanks on
board":GOTO 1300
1330 T=T+A:A(S)=A(S)-A
1380 GOTO 110
1390 REM End of game
1400 PRINT:PRINT "You used";N;"tanks of fuel, ";
1410 PRINT"brought";T
1420 A=0:FOR I=2 TO 9:A=A+A(I):NEXT I
1430 PRINT "with you and left";A;"in the desert."
1440 PRINT:PRINT"Type RUN to try again"
1450 END
```

ADRIAN MORGAN

MINESHAFT MANIA

Manic Miner, written by Matthew Smith, has achieved cult status in the world of computer games. Its combination of quirky humour and off-beat graphics proved an instant winner, and its central character, Miner Willy, looks set to star in a range of follow-up games.

Manic Miner, available on the 48 Kbytes ZX Spectrum and the Commodore 64, is fundamentally a very simple game that is based on an earlier best-seller called Kong. The object of that game was to climb up ladders and branches, all the while avoiding obstacles, in an attempt to rescue the distressed damsel held captive by the Great Ape. In Manic Miner, you take the role of Miner Willy, a prospector from that well-known mining centre, Surbiton. Willy stumbles upon a forgotten mineshaft in which a lost civilisation mined gold and other valuables. Unfortunately, the mine's former inhabitants forgot to disable the Manic Mining Robots, thus making the job of treasure retrieval extremely difficult.

There are 20 caverns in the mine, and in each of these there are four keys that must be procured before Willy can unlock the door leading to the next stage. Each cave has a number of different ledges onto which Willy must jump in order to reach the keys. Some of these ledges are rather weak (presumably with age) and give way as Willy reaches them. The caves are heralded by phrases like 'Eugene's Lair' (a reference to rival whizz-kid programmer Eugene Evans of Imagine), 'Miner Willy Meets the King Beast', 'Attack of the Mutant Telephones' (another 'in-joke', this time directed at programmer Jeff Minter's obsession with mutant llamas) and 'Skylab Landing Bay'. All of the caverns are inhabited by numerous alien beings whose very touch is instant death. Even the plants are lethal.

You help Willy to avoid all these problems with three simple commands: 'Left', 'Right' and 'Jump'. This is part of the game's attraction — the simplicity of the controls means that there is no long learning period, and you may select the keys you feel most comfortable with.

Willy has three lives, and in each incarnation he has a limited supply of air, indicated by an on-screen meter. As the loss of the third life takes Willy back to Cave One, the game can get very frustrating, and it is hardly surprising that some people have managed to rig the action so that they can start at any chosen cavern.

The Commodore translation is almost an exact copy of the Spectrum version and fails to take advantage of the 64's more versatile sound commands and higher resolution graphics. The playing area on the 64 has been made considerably smaller than the available screen size, so that it exactly matches the Spectrum version.

But both versions are undeniably great fun to play. The pace of the game and the difficulty of the problems posed have been carefully worked out, making it extremely addictive. And Matthew Smith has now produced a sequel, Jet Set Willy, which is rapidly creating a cult of its own...

Manic Miner: For 48K Spectrum, £5.95
For Commodore 64, £7.95

Publishers: Software Projects, Bear Brand Complex, Allerton Road, Woolton, Liverpool, Merseyside L25 7SF

Author: Matthew Smith

Joysticks: Both versions

Format: Cassette

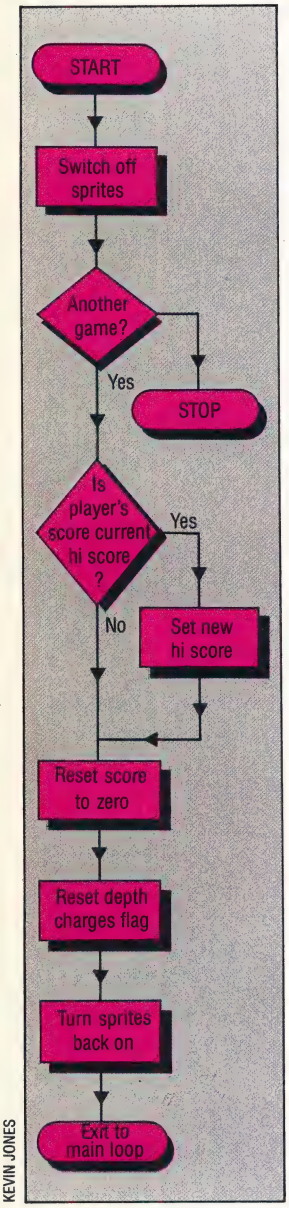
Manic Miner on the Spectrum

Manic Miner on the Commodore 64

Going Underground
The weird and wonderful objects that are to be found in Manic Miner's underworld have contributed to the game's cult status. Seasoned players often boast about the more unusual things they discover in the caverns

IAN McKINNELL

RUN SILENT, RUN DEEP



KEVIN JONES

At last we can apply the finishing touches to our Subhunter game. We set up the routines that create an explosion when a depth charge hits a submarine, and explain the end of game procedure.

In the penultimate instalment of the course, we discovered how easy it is to detect collisions between sprites using a sprite collision register, V+30. When this happens, the Hit subroutine (starting at line 5000) has three tasks to perform. First of all, it must cause an explosion at the point on the screen where the two sprites collided, and then it must increase the player's score by the value of the sub, which is calculated from its speed (DX) and its depth (Y3). Finally, it must reset the co-ordinates for the next sub to start moving across the screen. Let's look at the code for the Hit subroutine (lines 5000–5250) in more detail.

Line 5010 POKEs a zero into the collision register V+30 to clear it. Commodore claims the sprite collision register clears itself once two sprites have passed over each other and are no longer in collision. Experience, however, shows that the register does not always clear itself quickly enough, causing unexpected effects such as explosions occurring for no reason. The solution is to clear the collision register manually after a collision. Once this has been done the explosion sprite can be positioned and turned on.

Line 5030 gives the explosion an X co-ordinate ten pixels to the right of that of the depth charge. This slight shift positions the explosion more centrally over the depth charges. As X2 takes its value from the ship's X co-ordinate (X0), its value has an upper limit of 245. This means that the maximum value of the explosion's X co-ordinate is 255. The Y co-ordinate for the explosion is taken directly from that of the submarine.

The explosion sprite has been designated as sprite 1. Line 5040 sets bit 1 of the register V+21 to one, turning on sprite 1 without disturbing the values of other bits within the register. At this point it is interesting to note that the explosion sprite will appear on top, or in front of, the sub and depth charge sprites. This is known as *sprite priority*, and it is governed by the simple rule that lower numbered sprites appear in front of higher numbered ones. It is no accident that the explosion was designated as sprite 1 and the depth charges and sub were designated 2 and 3 respectively.

The colour of the explosion sprite is controlled by location V+40 of the VIC chip. An interesting effect can be obtained by rapidly changing the colour of the explosion using a FOR... NEXT loop to POKE in colour code numbers between 1 and 15. An outer FOR... NEXT loop repeats this process 20 times (lines 5060–5100). When the explosion is complete, all three sprites (explosion, depth charges and submarine) must disappear from the screen. Line 5130 turns sprites 1, 2 and 3 off.

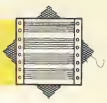
As mentioned previously, the player's score needs updating using the subroutine beginning at line 5500. As the score is to be increased by the sub's value (rather than decreased, as happens when a sub reaches the right hand side of the screen unscathed) the value of DS is set to one to signal this. Finally, before another sub can travel across the screen, its co-ordinates need to be reset using the subroutine at 2500 and the sub sprite must be turned back on. In addition, the flag that signals the dropping of a depth charge must be reset to zero so that the player can start firing depth charges again.

At the end of three minutes the program leaves the main loop and jumps to line 400. When we first discussed the use of the Commodore 64 timer (see page 234) line 400 was a simple END statement. The End of Game routine allows the game to be replayed and the highest scores recorded. The flowchart shows the tasks to be incorporated into such a routine. Lines 400 to 660 of the program listing perform these tasks. Most of the code is self-explanatory, remembering that CHR$(19) homes the cursor to the top left corner of the screen and CHR$(144) causes subsequent PRINTed letters to be coloured black.

In this short programming project for the Commodore 64 we have learned how to construct a simple animated game. In building up the program we have covered all the main aspects of programming this kind of game in BASIC. You may well wish to add refinements of your own to the program using the principles we have learned. One way of extending the game to make it more interesting would be to allow more activity on the screen by incorporating the four unused sprites.

A Table Of The Variables Used In Subhunter

V	Start of the VIC chip registers
FL	Depth charges flag – set to one if a charge is dropped
SC	Current player's score
HS	Highest score so far
TI$	Commodore 64's own timer
X0	X co-ordinate of ship
X2,Y2	X and Y co-ordinates of depth charge
X3,Y3	X and Y co-ordinates of sub
H3,L3	Hi byte and lo byte of sub's X co-ordinate
DX	Number of pixels by which X co-ordinate of sub is increased
DS	Flags whether a score is to be increased (DS=1) or decreased (DS=–1)



Subhunter – The Final Listing

```
10 REM ****************************
30 REM ** 64 PROGRAMMING PROJECT **
70 REM ****************************
90 POKE55,0:POKE56,48:CLR:
REM LOWER MEMTOP
100 V=53248:FL=0:SC=0
110 GOSUB1000: REM SCREEN SETUP
120 GOSUB2000: REM SPRITE CREATION
130 GOSUB2500: REM SET SUB COORDS
140 TI$="000000"
200 REM **** MAIN LOOP ****
210 REM   ** TIMER **
220 PRINTCHR$(19);:PRINTTAB(14)CHR$(5);
"TIME "MID$(TI$,3,2)":"RIGHT$(TI$,2)
225 IFVAL(TI$)>259 THEN 400:REM END GAME
230 GET A$
240 IF A$="Z" THEN XO=XO-1.5:IF XO<24
THENXO=24
250 IF A$="X" THEN XO=XO+1.5:IF XO>245
THENXO=245
260 IF A$="M" AND FL=0 THEN GOSUB3000:
REM SET UP DEPTH CHARGES
270 REM  ** MOVE SHIP **
290 POKE V,XO
300 REM ** MOVE SUB **
310 X3=X3+DX
320 REM**IF SUB REACH EDGE OF SCREEN **
330 IF X3>360 THEN DS=-1:GOSUB5500:
GOSUB2500
340 H3=INT(X3/256):L3=X3-256*H3
350 POKE V+6,L3
360 IF H3=1 THEN POKE V+16,PEEK(V+16)OR8:
GOTO380
370 POKE V+16,PEEK(V+16)AND247
380 IF FL=1 THEN GOSUB4000:
REM MOVE DEPTH CHARGE
390 GOTO 200:REM RESTART MAIN LOOP
400 REM **** END OF GAME CONDITIONS ****
410 REM   ** TURN OFF SPRITES **
420 POKE V+21,0
430 REM   ** RESET SUB & SHIP COORDS **
440 XO=160:GOSUB 2500
450 INPUT"ANOTHER GAME (Y/N)";AN$
460 IF AN$<>"Y"THENEND
480 REM   ** RUBOUT MESSAGE **
490 PRINT CHR$(19):REM HOME CURSOR
500 FOR I=1 TO 120
510 PRINT" ";
520 NEXT I
540 REM   ** SET HI SCORE **
550 IF SC>HS THEN HS=SC
560 PRINT CHR$(19);CHR$(144);" SCORE 000":
SC=0
570 PRINT CHR$(19);
580 PRINT TAB(26);CHR$(144);"HI SCORE";HS
600 REM   ** RESET TIMER AND FLAG **
610 TI$="000000":FL=0
630 REM   ** TURN ON SUB & SHIP **
640 POKE V+21,9
660 GOTO200: REM RESTART LOOP
1000 REM **** SCREEN SETUP ****
1010 PRINT CHR$(147):REM CLEAR SCREEN
1030 REM   ** COLOUR SEA **
1040 POKE 53281,14:POKE 53280,6
1050 FOR I=1264 TO 1943
1060 POKE I,160:POKE I+54272,6
1070 NEXT
1090 REM   ** SEA BOTTOM **
1100 FORI=1944 TO 2023
1110 POKE I,102:POKE I+54272,9
1120 NEXT
1130 POKE 650,128:REM REPEAT KEYS
1150 REM   ** SCORE **
1160 PRINT CHR$(19);CHR$(144);" SCORE 000"
;SPC(16);"HI SCORE 000"
1170 RETURN
2000 REM **** SPRITE CREATION ****
2020 REM   ** READ SHIP DATA **
2030 FOR I= 12288 TO 12350
2040 READ A:POKE I,A:NEXT I
2060 REM   ** READ SUB DATA **
2070 FOR I=12352 TO 12414
2080 READ A:POKE I,A:NEXT
2100 REM   ** READ CHARGES DATA **
2110 FOR I = 12416 TO 12478
2120 READ A:POKE I,A:NEXT
2140 REM   ** READ EXPLOSION DATA **
2150 FOR I = 12480 TO 12542
2160 READ A:POKE I,A:NEXT
2180 REM   ** SET POINTERS **
2190 POKE 2040,192:POKE 2041,193:POKE 2042
,194
2200 POKE2043,195
2220 REM   ** SET COLOURS **
2230 POKE V+39,0:POKE V+40,1:POKE V+41,0
2240 POKE V+42,0
2260 REM   ** SET INITIAL COORDS **
2270 POKE V+1,80:XO=160: REM SHIP COORDS
2280 POKE V+29,15:POKE V+23,2
2300 REM   ** TURN ON SPRITES 0 & 3 **
2310 POKE V+21,9
2320 RETURN
2500 REM **** RESET SUB COORDS ****
2510 Y3=110+INT(RND(TI)*105)
2520 POKE V+7,Y3:POKE V+6,0
2530 X3=0:DX=RND(TI)*3+1
2540 POKE V+16,0
2550 RETURN
3000 REM **** SETUP DEPTH CHARGES ****
3020 REM   ** SET FLAG **
3030 FL=1
3050 REM   ** SET COORDS **
3060 Y2=95:X2=XO
3070 POKE V+4,X2:POKE V+5,Y2
3090 REM   ** TURN ON SPRITE 2 **
3100 POKE V+21,PEEK(V+21)OR4
3110 RETURN
4000 REM **** MOVE DEPTH CHARGE ****
4020 REM   ** DECREASE Y COORD **
4030 Y2=Y2+2
4050 REM   ** TEST SEA BTM & TURN OFF **
4060 IF Y2>Y3+25 OR Y2>216 THEN POKEV+21,
PEEK(V+21)AND251:FL=0
4070 POKE V+5,Y2
4090 REM   ** TEST FOR HIT ON SUB **
4100 IF PEEK(V+30)=12 THEN GOSUB 5000:
REM HIT ROUTINE
4110 RETURN
5000 REM **** HIT ROUTINE ****
5010 POKE V+30,0:REM CLR COLLISION REG.
5020 REM   ** TURN ON EXPLOSION SPRITE **
5030 POKE V+2,X2+10:POKE V+3,Y3
5040 POKE V+21,PEEK(V+21)OR2
5060 REM   ** FLASH COLOURS **
5070 FOR I=1 TO 20
5080 FOR J=1TO 15
5090 POKE V+40,J
5100 NEXT J:NEXT I
5120 REM   ** TURN OFF SPRITES 1,2 & 3**
5130 POKE V+21,PEEK(V+21)AND241
5150 REM   ** UPDATE SCORE **
5160 DS=1:GOSUB 5500
5180 REM   ** RESET SUB COORDS & FLAG **
5190 FL=0:GOSUB 2500
5210 REM   ** TURN SUB BACK ON **
5220 POKE V+21,PEEK(V+21)OR8
5230 RETURN
5500 REM **** UPDATE SCORE ****
5510 SC=SC+INT(Y3+DX*30)*DS
5520 IF SC<0 THEN SC=0
5530 PRINT CHR$(19);CHR$(144)" SCORE";SC;
CHR$(157);"     "
5540 RETURN
6000 REM **** SHIP DATA ****
6010 DATA0,0,0,0,0,0,0,0
6020 DATA0,128,0,0,192,0,0,192,0
6030 DATA0,192,0,1,224,0,1,224,0
6040 DATA13,224,0,3,248,128,3,253,8
6050 DATA15,254,16,31,255,48,255,255,255
6060 DATA127,255,254,63,255,254,31,255,252
6070 DATA0,0,0,0,0,0,0,0
6100 REM **** EXPLOSION DATA ****
6110 DATA0,0,0,0,0,0,0,16,0,0,8,0,4,16
6120 DATA0,3,2,64,1,56,128,12,255,144
6130 DATA1,238,40,5,151,0,11,121,0,1
6140 DATA183,0,25,214,96,0,236,48,6,24
6150 DATA152,3,98,0,8,51,0,0,96,128,0
6160 DATA64,0,0,0,0,0,0,0
6170 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6200 REM **** DEPTH CHARGES DATA ****
6210 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0
6220 DATA0,0,0,32,0,0,32,0,0,32,0,0,32,0
6230 DATA0,0,0,0,0,0,0
6240 DATA2,0,0,2,0,0,2,0,0,2,0,0
6250 DATA0,0,0,0,0,0,0,0
6260 DATA0,0,0,0,0,0,0,0
6300 REM **** SUBMARINE DATA ****
6310 DATA0,0,0,0,0,0,0,0,0,0,0,0
6320 DATA0,8,0,0,12,0,0,12,0
6330 DATA0,12,0,0,28,0,0,60,0
6340 DATA0,126,0,199,255,255
6350 DATA239,255,255,127,255,255
6360 DATA255,255,254,199,255,254
6370 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Here then is the final listing for our Subhunter program together with a table of the key variables used in it. The listing contains many REM statements to aid understanding. These may be left out when typing the program into your own computer, but be careful that you do not delete a REM line that is required by another part of the program. For example, you may choose to delete the REM at line 400, but this line number is used as part of a GOTO statement in line 225. Deleting line 400 entirely will cause an 'UNDEF'D STATEMENT ERROR AT LINE 225' message to appear and the program will crash. The best way to avoid this is to leave out only those REMs that appear at the end of a line and those lines that use colons (:) to space out the code.

THE GREAT DIVIDE

We conclude this series of machine code tutorials with a brief study of unsigned binary division and the use of operating system ROM routines in Assembly language screen display programming. In a summary of this introductory section of the course, we review the major themes and topics — from BASIC to branching, from arrays to assemblers.

Just as we used the manual long multiplication method as an algorithm for binary multiplication (see page 298), so the manual long division method is a model for binary division. Consider this binary long division:

```
     00001110 r00          quotient
1011)10011010              dividend
   -1011                   subtract divisor
    10000
    -1011                  subtract divisor
     1011
     -1011                 subtract divisor
        00                 no remainder
```

The essence of the method is the repeated subtraction of the divisor from the high order bits of the dividend. Depending on the result of this subtraction, a zero or a one is shifted into the quotient. The remainder is the result of the last subtraction of a divisor.

The various ways in which this algorithm may be implemented in Assembly language are not as apparent as they were for multiplication. However, as before, the Z80 version uses the power and flexibility of its 16-bit registers, while the 6502 must fetch and carry eight bits at a time. The divisor is in the address labelled DIVSR, the dividend in DVDND, the quotient in QUOT, and the remainder in RMNDR. The program in Z80 and 6502 Assembly language is given.

Notice in both cases that when the divisor is subtracted from the partial dividend with a negative result, the dividend must be restored by adding the divisor back in again. The 6502 version is noteworthy for its treatment of the processor status register after the divisor subtraction: the carry flag must be rotated into the quotient, but its state must also be preserved to indicate the result of the subtraction. Consequently, the PSR is pushed onto the stack before the rotation, and pulled off it afterwards, thus restoring the carry to its immediate post-subtraction state.

We have now considered the four rules of arithmetic — this is plainly worth doing as a programming exercise for the insight it brings to machine processes, but inventing all the various combinations of single- and multiple-byte arithmetic is unnecessary, given that programmers have been writing these routines in textbooks and magazines for years. When the need arises for variations of the routines that we have developed, they will be supplied or set as exercises.

16-BIT BY 8-BIT DIVISION

Z80			6502		
START	LD	A,(DIVSR)	START	LDA	#$00
	LD	D,A		STA	QUOT
	LD	E,$00		STA	RMNDR
	LD	HL,(DVDND)		LDX	#$08
	LD	B,$08		LDA	DVDHI
LOOP0	AND	A		SEC	
	SBC	HL,DE		SBC	DIVSR
	INC	HL	LOOP0	PHP	
	JP	P,POSRES		ROL	QUOT
NEGRES	ADD	HL,DE		ASL	DVDLO
	DEC	HL		ROL	A
POSRES	ADD	HL,HL		PLP	
	DJNZ	LOOP0		BCC	CONT1
	LD	(QUOT),HL		SBC	DIVSR
	RET			JMP	CONT2
DIVSR	DB	$F9	CONT1	ADC	DIVSR
DVDND	DW	$FDE8	CONT2	DEX	
QUOT	DB	$00		BNE	LOOP0
RMNDR	DB	$00		BCS	EXIT
				ADC	DIVSR
				CLC	
			EXIT	ROL	QUOT
				STA	RMNDR
				RTS	
			DIVSR	DB	$F9
			DVDLO	DB	$E8
			DVDHI	DB	$FD
			QUOT	DB	$00
			RMNDR	DB	$00

SCREEN OUTPUT

So far in the course we have used RAM memory and the CPU as a calculating system, and left the results of our efforts somewhere in RAM to be inspected manually using a monitor program. This is obviously unsatisfactory, but until arithmetic and subroutine calls had been studied there was simply no point in considering the screen output from machine code.

Most micros have a memory-mapped display. This means that an area of RAM is dedicated to holding an image of the screen. The screen display is composed of dots, or pixels, which are either on or off. These can, therefore, be represented by

binary ones (on) or zeros (off), and the entire contents of the screen can be regarded as a 'mapping' into dots of the bits that comprise those bytes of screen RAM. Unfortunately, although the BBC Micro, the Spectrum and the Commodore 64 all use this mapping technique, none of them does so in a straightforward manner. For our purposes, the simplest method would be to divide each row of the screen into pixel bytes numbered consecutively from left to right, the leftmost byte in a row following the rightmost in the preceding row. For a variety of reasons this is not the case on any of these machines. Let's consider each case separately.

The Spectrum screen is always in high resolution mode, and a fixed area of memory is set aside for mapping the screen. The mapping is complex, however, as the screen is divided horizontally into three blocks of eight PRINT rows, and each print row is divided horizontally into eight pixel rows. The addressing of the bytes that comprise these rows is sequential within the rows, but not between the rows. The BBC Micro and the Commodore 64 do not follow this pattern, but are equally devious. For the moment, it is considerably easier to understand if we confine ourselves to outputting ASCII characters to the screen.

This is something that the machine does all the time, and there are, therefore, machine code routines in ROM for the purpose. Given a suitably detailed description of their operation, we can call these routines from our own Assembly language programs. What we need to know is the call address, the communication registers, and any necessary preliminaries.

On the Spectrum there are no preliminaries to observe, and the communicating register is the accumulator, which must contain the ASCII code of the character to be printed. We need only issue the instruction RST $10 and the character whose code is in the accumulator will be printed on the screen at the current cursor position. This is very much the pattern of the other two systems, but the RST (ReSTart) op-code is peculiar to the Z80 command set: it is a single-byte zero-page branch instruction that must take one of only eight possible operands—$00,$08,$10,$18, etc. to $38. Each of these locations points to the start address of a ROM routine, somewhere in zero page. These routines are typically dedicated to handling input and output, and we call them through the RST instruction rather than directly by address. This is partly for speed (it is quicker to use RST than CALL, although only the CPU would notice the difference), and partly for the sake of the program's portability. If every Z80 programmer knows that RST $10 calls the PRINT routine on every Z80 machine, then nobody is going to bother about where a particular systems software engineer actually locates the PRINT routine, and the engineer is free to locate it anywhere, provided that zero page is arranged in such a way that the RST locations direct programs to the start addresses of the commonly-agreed routines.

On the BBC Micro the procedure is similar: an ASCII code in the accumulator combined with a JSR $FFEE command will cause the character to be PRINTed on the screen at the current cursor position. This is the OSWRCH routine, much referred to in BBC literature and well documented in the Advanced User Guide.

The Commodore 64 follows the pattern of the other two machines. An ASCII code in the accumulator and a JSR $FFD2 command causes the character to be PRINTed at the current cursor position. This is the CHKOUT routine, and is documented in the Programmer's Reference Guide.

This, therefore, is the general pattern of use of ROM routines and demonstrates the principle of communication registers. A communication between the calling program and a subroutine may pass either way — an input routine, for example, might pass a character from an external device to the CPU via the accumulator. Even when there is no substantive information passed like this, an error code may well be returned from the subroutine through one of the registers. This sort of protocol is documented in the many machine-specific works of reference now available.

Input from the keyboard and other devices will be dealt with in later instalments, as will high resolution plotting from machine code. We conclude this instalment of the course with a summary of the various aspects of Assembly language and machine code programming.

IN SUMMARY

We began the course with a wide-ranging look at machine code from a very non-specific point of view, trying to dispel some of its mystique and place it in context as just one kind of code among all the others that we (and computers) use. We have seen how the same sequence of bytes in RAM can be interpreted at one moment as a string of ASCII data, at the next as a BASIC program line, at the next as a string of two-byte addresses, and then again as a sequence of machine code instructions. A few minutes spent playing with a machine code monitor program should convince you that some sequences of bytes can be disassembled as three quite different, but valid, sequences of instructions — depending on whether you start the disassembly at the first, second or third byte in the sequence. Nothing intrinsic to the code prevents this happening, and the CPU itself cannot tell whether it's executing the code that you wrote, or some garbled version of it, accidentally transposed in memory.

We went on to consider the organisation of memory, and the common conventions of addressing. To make any sense of this we had to begin the study of binary arithmetic, which immediately delineated the horizons on our view from the CPU — in eight-bit processors we are confined, except in particular circumstances, to

the limits of a byte (in other words, the range of decimal numbers 0-255). Once we encountered the meaning and appropriateness of binary arithmetic, the limitations of the decimal system for dealing with the world of Assembly language became apparent. In exploring the idea of paged memory we saw how the size of the logical pages must be a function of the number base, and in a binary system that means that the page size must be a power of two. Two to the power of eight gives 256 — the magic number in an eight-bit microprocessor system.

Binary very quickly became too unwieldy and too prone to error for use as a numbering system, and we passed on to hexadecimal (number base 16) arithmetic. We saw how the eight-bit byte can be fully represented by two hex digits, from $00 to $FF, one digit representing the state of the lower four bits, and the other standing for the upper four bits of the byte.

The way that BASIC programs are stored in the program area was exhaustively examined. By describing tokenisation as another form of machine code, we gave a useful insight into the operating system. Our discussion of end-of-line markers showed how the BASIC interpreter handles the difficulty of telling where one piece of code ends and another starts, and the Commodore's link addressing introduced both the lo-hi address convention and the idea of indirect addressing.

From there we moved directly into Assembly language itself. We started from the primitive operations of the CPU as directed by the eight-bit op-codes that constitute its program instructions. With the idea of coding so thoroughly explored, it was a short step to Assembly language mnemonics. Once we had made that step it became clearer that programming in machine code or Assembly language or BASIC was still just programming, and that what counted was solving the logical problem before worrying about how to code the solution. Problem-solving has been the central theme of the course. But the obscurity of some of Assembly language's concepts forced our attention first to clearing the haze of confusion that besets most people on first contact with low-level languages.

The course proceeded to spend some time on the practicalities of loading and running machine code programs on computers that were more or less dedicated to running BASIC programs. We looked at system variables and operating system pointers on the BBC Micro, Spectrum, and Commodore 64, and learned how to 'steal' space from BASIC.

We glanced at the architecture of small computer systems and the Z80 and 6502 CPUs, and moved on to begin writing Assembly language programs that manipulated memory and the accumulator. Assembler directives or pseudo-ops were introduced here, a step towards practicality and the real world, but also a step away from machine code, manual assembly, and the laborious detail of low-level programming.

The need for the logical constructs of a programming language was now obvious, and we turned to considering the processor status register (PSR). Its role as a recorder of the results of CPU operations was immediately illustrated in an introduction to binary arithmetic, using the 'add with carry' instruction. The central role of the PSR and, in arithmetic, of the carry flag, was obvious as soon as it was seen. The course has concentrated on the processor status register and the associated instructions since then.

We briefly examined the various addressing modes; indexed addressing was given most attention because of its importance in handling loops, lists and tables. The need for a class of instruction to change the flow of control in a program is evident once these structures are introduced, so we began to examine the conditional branch instructions while still exploring the potential of indexed and indirect addressing. With conditional branching, primitive arithmetic and array-type structures, we have almost all the bones of any programming language. Fleshing out the form through practice and systematic investigation is the remaining task.

The Assembly language subroutine call and return was examined both for itself and as a way of introducing the last unexplored area of the operating system — the stack. Seeing how it works, what it is for, and how we might use it introduced some new ploys to the repertoire of machine code programming, while a more searching look at the CPU registers and their interactions introduced new possibilities in the manipulation of memory and the microprocessor.

Finally, with a working knowledge of the architecture of the microprocessor and a vocabulary of op-code instructions, we approached binary arithmetic. The oddities of subtraction and two's complement, and the complexities of multiplication and division have all been covered in detail. Looking ahead, we will investigate the practical craft of machine code programming by investigating and exploring specific tasks for the processors we have initially concentrated our attention on (the Z80 and the 6502), as well as other processors, such as the 6809 CPU used by the Dragon 32 and 64.

Answers To Exercises On Page 299

1) The fastest-running solution is certainly a routine written specifically for 16-bit multiplicands, on the same lines as the eight-bit routine in the last instalment. On the other hand, if you split 16-bit multiplication into two separate eight-bit multiplications (multiplier by lo-byte, followed by multiplier by hi-byte), then you can call the existing eight-bit routine twice, adjust for a carry out of the lo-byte, and store the results in the product bytes.

2) A multiplication routine using repeated addition consists simply of a loop whose counter is the value of the multiplier; each time the loop is executed, the multiplicand is added into the product.

ROR — ROTATE RIGHT

Register — 6A (1 byte)

6502

The contents of the byte are rotated one bit right through the carry.

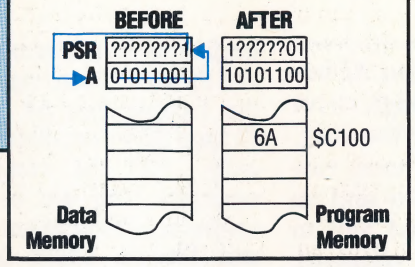
EFFECT ON PSR

	S	V		B	D	I	Z	C
MSB	X						X	X

LSB

Example:

LOCATION	MACHINE CODE	ASSEMBLY LANGUAGE
C100	6A	ROR A



RR — ROTATE RIGHT

Register CB (2 bytes)

Z80

The contents of the byte are rotated one bit right through the carry.

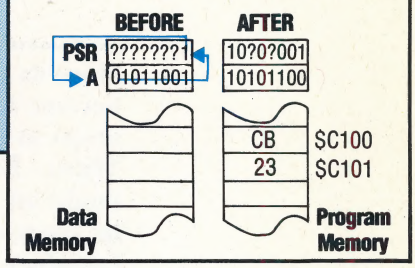
EFFECT ON PSR

	S	Z		H		V	N	C
MSB	X	X		0		X	0	X

LSB

Example:

LOCATION	MACHINE CODE	ASSEMBLY LANGUAGE
C100	CB 1C	RR H



SBC — SUBTRACT WITH BORROW

Absolute - ED (3 bytes)

6502

The contents of the memory location are subtracted from the accumulator; carry shows borrow status.

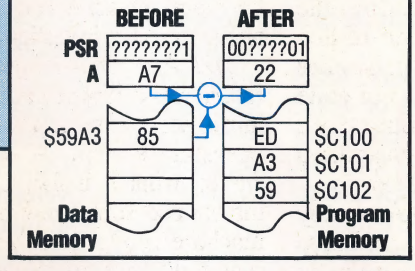
EFFECT ON PSR

	S	V		B	D	I	Z	C
MSB	X	X					X	X

LSB

Example:

LOCATION	MACHINE CODE	ASSEMBLY LANGUAGE
C100	ED A3 59	SBC $59A3



SBC — SUBTRACT WITH BORROW

Immediate — DE (2 bytes)

Z80

The contents of the byte following the op-codes are subtracted from the accumulator.

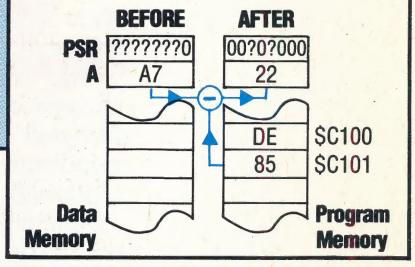
EFFECT ON PSR

	S	Z		H		V	N	C
MSB	X	X		X			X	X

LSB

Example:

LOCATION	MACHINE CODE	ASSEMBLY LANGUAGE
C100	DE 85	SBC A,$85



ASL — ARITHMETIC SHIFT LEFT

Register — 0A (1 byte)

6502

The contents of the byte are shifted one bit left through the carry; zero is shifted into lsb.

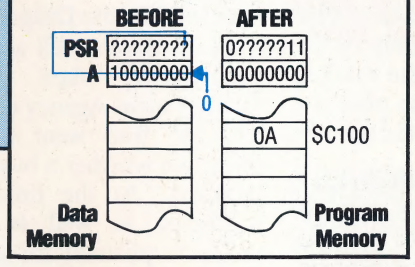
EFFECT ON PSR

	S	V		B	D	I	Z	C
MSB	X						X	X

LSB

Example:

LOCATION	MACHINE CODE	ASSEMBLY LANGUAGE
C100	0A	ASL A



SLA — SHIFT LEFT ARITHMETIC

Register — CB (2 bytes)

Z80

The contents of the byte are shifted one bit left through the carry; zero is shifted into lsb.

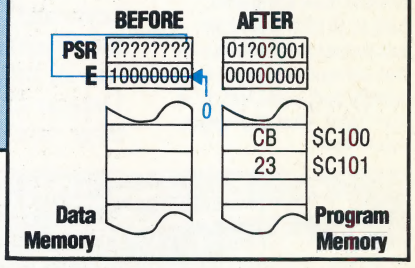
EFFECT ON PSR

	S	Z		H		V	N	C
MSB	X	X		0		X	0	X

LSB

Example:

LOCATION	MACHINE CODE	ASSEMBLY LANGUAGE
C100	CB 23	SLA E



CMP — COMPARE THE ACCUMULATOR

Absolute - CD (3 bytes)

6502

The contents of the memory location are compared with the accumulator.

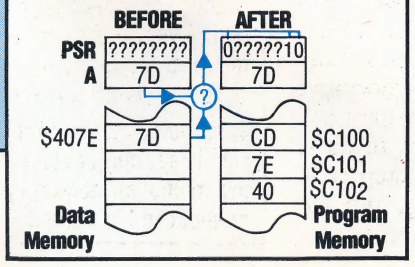
EFFECT ON PSR

	S	V		B	D	I	Z	C
MSB	X						X	X

LSB

Example:

LOCATION	MACHINE CODE	ASSEMBLY LANGUAGE
C100	CD 7E 40	CMP $407E



CP — COMPARE THE ACCUMULATOR

Immediate — FE (2 bytes)

Z80

The contents of the byte following the op-code are compared with the accumulator.

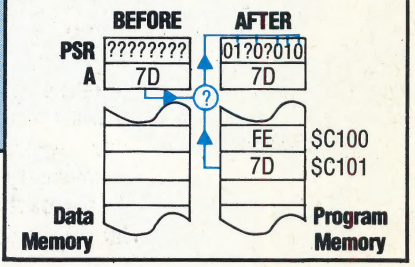
EFFECT ON PSR

	S	Z		H		V	N	C
MSB	X	X		X		X	1	X

LSB

Example:

LOCATION	MACHINE CODE	ASSEMBLY LANGUAGE
C100	FE 7D	CP $7D



WELSH RARE BIT

Since its launch in 1982, the Dragon 32 has become as familiar a sight in the high street shops as the Sinclair Spectrum or the BBC Micro. But financial problems have cast doubt on the future of the company and on its plans to market the Dragon 64 and to introduce a micro to MSX standard.

Dragon Data was first established as a subsidiary of Mettoy, the toy manufacturers, in 1981. Mettoy's intention was to cash in on the boom in home computers, then just beginning in the United Kingdom. With financial assistance from the Welsh Development Agency, a factory was set up in Swansea, and the Dragon 32 made its first appearance in August 1982.

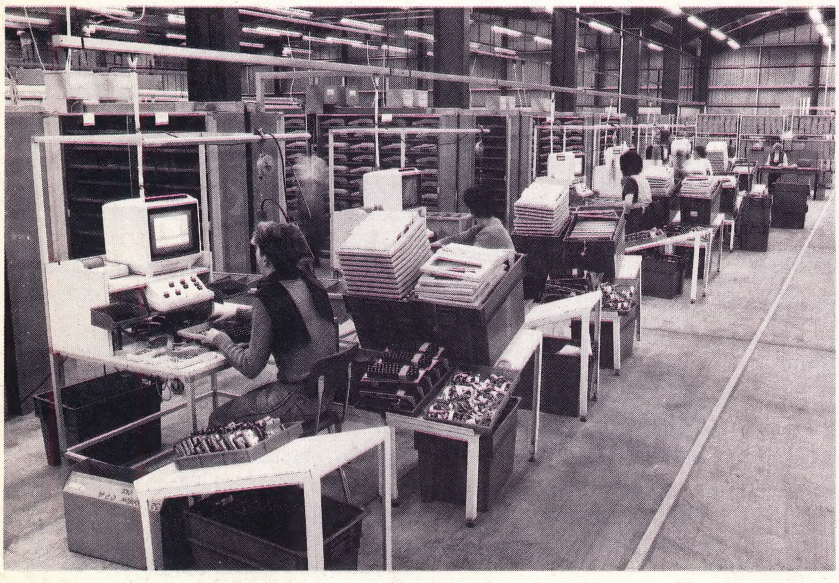
The company opted for Motorola's 6809 microprocessor, rather than the Z80 or 6502 favoured by most other home computer manufacturers. The Dragon's circuitry followed Motorola's recommended layout, which led to accusations that Dragon Data had based its design on Tandy's Color computer, another model that used the Motorola format. A side effect of this was that users soon discovered that some software written for the 'CoCo' would run on the Welsh machine as well.

The Dragon 32's major selling points were its Microsoft BASIC (the most widely used BASIC dialect) and its full-sized typewriter-style keyboard. At the time the machine was launched, the Dragon's keyboard was matched only by the Vic-20 in the under-£200 sector of the market. Dragon Data's marketing strategy also played a large part in the machine's success; in the months leading up to Christmas 1982 the ZX Spectrum and the BBC Micro were both in short supply, and the Commodore 64 had yet to appear. The Dragon 32 was available in large numbers, and by early 1983 the company had sold 32,000 machines. This was in part due to the Mettoy connection; those major chain stores such as Boots and Dixons, which had always stocked the company's toys, were more than happy to sell the new computer.

However, in the summer of 1983, Dragon Data found itself in deep financial trouble. The company was expanding when Mettoy went into receivership, casting doubt on the future of its Welsh subsidiary. Dragon was eventually saved by a consortium of companies led by Prutec, the high technology investment arm of the giant Prudential insurance company. A £2.5 million rescue package was put together, and the firm acquired a new managing director in Brian Moore, a former executive of GEC. These changes enabled Dragon Data to overcome its cash flow problems, to invest in a new manufacturing plant in Port Talbot, and to continue its development of the Dragon 64 and disk drive.

The Dragon 64 has 64 Kbytes of RAM, an improved keyboard and an RS232C serial interface. The disk drive uses standard 5¼" floppy disks running under Dragon DOS, which can be used by both the 32 and 64 models. A version of the powerful OS9 operating system is also available for the Dragon 64.

But a shadow fell across all Dragon's plans in June 1984, when Prutec and the Welsh Development Agency refused to put up more cash and the firm went into receivership. It was uncertain whether a buyer could be found for the company. At the time there were three new machines, as well as other computer-related products planned for 1984 alone. One of the planned micros was intended to meet the MSX standard being introduced by Japanese companies (see page 141). But the Welsh Dragon had become an endangered species.



Birth Of A Dragon
Dragon Data is unusual among British home computer companies in that it builds its own machines, whereas most companies, such as Sinclair and Acorn, subcontract the production of their computers. Dragon have recently built a new factory at Port Talbot, West Glamorgan, shown here

Market Wizard
Richard Wadman, the Marketing Director of Dragon Data, was planning to sell a whole new range of Dragon computers when the company suddenly went into receivership

Mentathlete

Home computers. Do they send your brain to sleep – or keep your mind on its toes?

At Sinclair, we're in no doubt. To us, a home computer is a mental gym, as important an aid to mental fitness as a set of weights to a body-builder.

Provided, of course, it offers a whole battery of genuine mental challenges.

The Spectrum does just that.

Its education programs turn boring chores into absorbing contests – not learning to spell 'acquiescent', but rescuing a princess from a sorcerer in colour, sound, and movement!

The arcade games would test an all-night arcade freak – they're very fast, very complex, very stimulating.

And the mind-stretchers are truly fiendish. Adventure games that very few people in the world have cracked. Chess to grand master standards. Flight simulation with a cockpit full of instruments operating independently. Genuine 3D computer design.

No other home computer in the world can match the Spectrum challenge – because no other computer has so much software of such outstanding quality to run.

For the Mentathletes of today and tomorrow, the Sinclair Spectrum is gym, apparatus and training schedule, in one neat package. And you can buy one for under £100.

sinclair

THE HOME COMPUTER ADVANCED COURSE

WE HAVE DESIGNED BINDERS SPECIALLY TO KEEP YOUR COPIES OF THE 'ADVANCED COURSE' IN GOOD ORDER.

All you have to do is complete the reply-paid order form opposite – tick the box and post the card today – **no stamp necessary!**

By choosing a standing order, you will be sent the first volume free along with the second binder for £3.95. The invoice for this amount will be with the binder. We will then send you your binders every twelve weeks – as you need them.

Important: This offer is open only whilst stocks last and only one free binder may be sent to each purchaser who places a Standing Order. Please allow 28 days for delivery.

Overseas readers: This free binder offer applies to readers in the UK, Eire and Australia only. Readers in Australia should complete the special loose insert in issue 1 and see additional binder information on the inside front cover. Readers in New Zealand and South Africa and some other countries can obtain binders now. For details please see inside the front cover. Binders may be subject to import duty and/or local tax.

The Orbis Guarantee: If you are not entirely satisfied you may return the binder(s) to us within 14 days and cancel your Standing Order. You are then under no obligation to pay and no further binders will be sent except upon request.

PLACE A STANDING ORDER TODAY.